НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

(повне найменування інституту, факультету)

Автоматизованих систем обробки інформації і управління

(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)
“ ” 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки ***6.050103 «Програмна інженерія»***

на тему *Бібліотека паралельних обчислень Петрі-об’єктних моделей*

Виконав: студент IV курсу, групи *ІП-51 Педоренко Андрій Вікторович*
(прізвище, ім’я, по батькові) (підпис)

Керівник *проф., д.т.н., Стеценко І.В*
посада,науковий ступінь,вчене звання,прізвище,ініціали (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.*
посада,науковий ступінь,вчене звання,прізвище,ініціали (підпис)

Рецензент:

проф. каф. ОТ, д.т.н., доцент Клименко І.А.
посада,науковий ступінь,вчене звання,прізвище,ініціали (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) – *6.050103 «Програмна інженерія» (Програмне забезпечення систем)*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *О.А. Павлов*
(підпис) (ініціали, прізвище)
“___”_____2019 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТУ

Педоренку Андрію Вікторовичу
(прізвище, ім’я, по батькові)

1. Тема проекту «*Бібліотека паралельних обчислень Петрі-об’єктних моделей*»
керівник проекту *Стеценко Інна Вячеславівна, д.т.н., проф.*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “23” квітня 2019 р. №1181-с
2. Термін подання студентом проекту «03» *червня 2019 року*

3. Вихідні дані до проекту
Технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення, архітектура програмного забезпечення
3) Аналіз якості програмного забезпечення та опис випробувань
4) Керівництво користувача, інтеграція бібліотеки

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна компонентів програмного забезпечення

3) Схема структурна класів програмного забезпечення

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «12» *березня 2018 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	*Вивчення рекомендованої літератури*	*28.04.19*	
2.	*Аналіз існуючих методів розв'язання задачі*	*28.04.19*	
3.	*Постановка та формалізація задачі*	*05.05.19*	
4.	*Аналіз вимог до програмного забезпечення*	*05.05.19*	
5.	*Алгоритмізація задачі*	*09.05.19*	
6.	*Моделювання програмного забезпечення*	*09.05.19*	
7.	*Обґрунтування використовуваних технічних засобів*	*10.05.19*	
8.	*Розробка архітектури програмного забезпечення*	*12.05.19*	
9.	*Розробка програмного забезпечення*	*19.05.19*	
10.	*Налагодження програми*	*26.05.19*	
11.	*Виконання графічних документів*	*28.05.19*	
12.	*Оформлення пояснювальної записки*	*28.05.19*	
13.	*Подання ДП на попередній захист*	*30.05.2019*	
14.	*Подання ДП рецензенту*		
15.	*Подання ДП на основний захист*	*07.06.2019*	

Студент _____ Педоренко А.В.
(підпис)

Керівник проекту _____ Стеценко І.В.
(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	КПІ.ІП-5117.045490.01.81	Пояснювальна записка	71	
3	А4	КПІ.ІП-5117.045490.02.13	Опис програми	13	
4	А4	КПІ.ІП-5117.045490.03.91	Технічне завдання	10	
5	А4	КПІ.ІП-5117.045490.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-5117.045490.05.34	Керівництво користувача	11	
7	А3	КПІ.ІП-5117.045490.06.99.СС	Схема структурна варіантів використань	1	
8	А3	КПІ.ІП-5117.045490.06.99.СС	Схема структурна компонентів програмного забезпечення	1	
9	А3	КПІ.ІП-5117.045490.06.99.СС	Схема структурна класів програмного забезпечення	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-5117.045490.00.90		
Розробн.		Педоренко А.В.			Відомість дипломного проекту	Літ.	Лист 1 / Листів 1
Керівн.		Стеценко І.В.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-51	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

АНОТАЦІЯ

Пояснювальна записка дипломного проекту складається з чотирьох розділів, містить 56 таблиць, 16 рисунків та 10 джерел — загалом 71 сторінка.

Об’єкт дослідження: паралельні алгоритми обчислення Петрі-об’єктних моделей.

Мета дипломного проекту: підвищення швидкодії алгоритму імітації Петрі- об’єктної моделі та представлення цього у вигляді лаконічної гнучкої бібліотеки.

У першому розділі виконано аналіз предметної області, відомих технічних рішень, сформульовано функціональні та нефункціональні вимоги до розроблюваного програмного забезпечення.

У другому розділі побудовано схеми BPMN для опису бізнес-процесів, розроблена архітектура програмного забезпечення та проаналізована структура вхідних та вихідних даних.

У третьому розділі проведено аналіз якості програмного забезпечення та наведено тестовий приклад.

У четвертому розділі описаний процес інтеграції бібліотеки та роботи з нею.

КЛЮЧОВІ СЛОВА: СТОХАСТИЧНІ МЕРЕЖІ ПЕТРІ, ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ, АЛГОРИТМ ІМІТАЦІЇ

ABSTRACT

Explanatory note of the diploma project consists of four sections, containing 56 tables, 16 figures and 10 sources - total 71 of pages.

The object of research: parallel algorithms for calculating Petri-object models.

The aim of the project: increase the speed of the simulation algorithm of the Petri-object model and present it in the form of a concise flexible library.

In the first section were analyzed subject area and known technical solutions, formulated functional and non-functional requirements for the software developed.

In the second section were constructed BPMN schemes for describing business processes, developed software architecture and analyzes the structure of input and output data.

The third section analyzes the software quality and provides a test case.

The fourth section describes the process of integrating and working with the library.

KEY WORDS: STOCHASTIC PETRI NET, PARALLEL CALCULATIONS, SIMULATION ALGORITHM

Пояснювальна записка
до дипломного проекту

на тему: Бібліотека паралельних обчислень Петрі-об’єктних моделей

Київ – 2019 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

BPMN — система умовних позначень (нотація) для моделювання бізнес-процесів.

JSON — відкритий стандарт формату файлу, який використовує текст, що може бути прочитаний людиною, для передачі даних у вигляді пар ключ-значення та масивів.

ВСТУП

Моделювання систем стає все більш затребуваним для бізнесу, так як витрати на її реалізацію зменшуються, а якість та інформативність отриманих результатів зростає. Замість того, щоб одразу налаштовувати процеси, бізнес аналізує їх характеристики та створює модель, за допомогою якої можна відслідкувати недоробку планів та знайти рішення для їх оптимізації.

Потужним інструментом моделювання є Петрі-об’єктні мережі, однак через їх сильну деталізацію розмір моделі швидко розростається, що призводить до підвищення часу роботи симуляції. Оскільки у комп’ютерах, на яких проводяться дослідження моделей все збільшується кількість центральних процесорів, розпаралелювання алгоритму симуляції призведе до значних заощаджень часу, витраченого на дослідження, а значить і витрат бізнесу на моделювання своїх процесів. Ця ідея призвела до створення даної роботи.

Мета даної роботи – підвищити швидкодію алгоритму імітації Петрі-об’єктної моделі.

Завданням даної роботи є розробка гнучкої бібліотеки обчислень Петрі-об’єктних моделей, яка будо симулювати моделі як послідовно, так і паралельно без потреби користувачеві перебудовувати моделі під алгоритм імітації. Результатом роботи є бібліотека мовою Java для її інтеграції в інші програмні продукти симуляції моделей розроблених.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Метою розробки паралельного алгоритму обчислень Петрі-об’єктних моделей є підвищення швидкодії алгоритму імітації Петрі-об’єктної моделі. У вигляді моделі з дискретним станом, який змінюється у часі під впливом подій можна представити велику кількість процесів із різних сфер, серед яких:

- виробництво;
- логістика;
- розробка транспортних шляхів;
- обслуговування клієнтів;
- громадський транспорт;
- комп’ютерні мережі;
- програмні додатки.

Представляють процеси у вигляді дискретно-подійних систем для того, щоб розробити, проконтролювати, проаналізувати або оптимізувати їх. Однак, для отримання точних даних системи повинні бути добре деталізованими та імітація має проводитися декілька разів, що створює високу складність обчислення та забирає багато часу. На великих системах один цикл імітації може займати декілька годин, або навіть декілька днів.

Дуже потужним засобом формалізацію дискретно-подійної системи є мережі Петрі. Найважливішою їх перевагою є гнучкість, що дозволяє створювати моделі систем, які неможливі в інших формалізмах. З іншого боку гнучкість призводить до сильної деталізації, а це ще більше підіймає складність обчислення. І хоча перехід по Петрі-об’єктної моделі покращує складність алгоритму [1], імітація все ще займає дуже багато часу. Для розв’язання цієї проблеми треба розробити паралельний алгоритм обчислення Петрі-об’єктної моделі.

1.2 Змістовний опис і аналіз предметної області

1.2.1 Підходи до паралелізму

Основна мета паралелізму — розбиття послідовної роботи алгоритму на менші частини, які можуть обчислюватися одночасно. Паралельний алгоритм імітації можна представити у вигляді декількох послідовних, які обмінюються між собою повідомленнями з вказаним часом їх виникнення.

Є декілька підходів до паралельної імітації дискретно-подійних моделей, які згідно з Джейсоном Лю [2] поділяються на чотири класи:

– тиражування випробувань: до цього типу паралелізму приписують одночасний запуск декількох спроб імітації моделі. Ефективність цього підходу висока, оскільки не потрібно синхронізуватися між процесами імітації. З іншого боку, мінімальний час обчислення обмежений часом обчислення однієї спроби;

– функціональна декомпозиція: цей підхід пропонує паралельно виконувати частини алгоритму імітації в межах однієї ітерації. Масштабованість цього підходу низька, оскільки різні частини однієї ітерації зазвичай сильно залежать одна від одної. Та все ж, в деяких моделях можна, наприклад, паралельно виконувати оновлення стану моделі;

– часова декомпозиція: цей підхід пропонує розділяти час імітації на менші інтервали, які імітуються паралельно, а потім глобально синхронізувати стан. Цей тип паралелізму добре масштабується, але його ефективність обмежена розміром інтервалу паралельної імітації та частотою й складністю синхронізації;

– просторова декомпозиція: в цьому випадку модель розбивають на менші частини, імітація на яких відбувається паралельно. Події, що відбуваються на межі частини, повинні оброблятися сусідньою частиною у відповідний час імітації, а конфлікти між частинами мають вирішуватися за допомогою синхронізації. У цього підходу великий потенціал, але його

ефективність залежить від вибору методу синхронізації та від природи моделі, що імітується.

Методи імітації просторово декомпонованих моделей можна розділити на синхронні та асинхронні. В синхронних методах події на межах синхронізуються в глобальні часові кроки, на яких всі потоки зупиняють свою імітацію та межовим станом частини моделі з сусідніми частинами моделі. Якщо часові кроки у всіх частин моделей рівні, то синхронізація виконується після кожного такого кроку. У випадку, коли час кроку частини відрізняється від часу глобального кроку, методи ігнорують синхронізацію проміжних інтервалів та синхронізуються в глобальні часові кроки. Зазвичай це призводить до чисельної помилки в обчисленні моделі. В асинхронних методах межові конфлікти вирішуються асинхронно та одночасно з локальною імітацією, під час якої сусідні частини моделі постійно обмінюються їх межовим станом. Отже, такі методи можна використовувати для точного паралельного імітування моделей, у яких у кожній частині моделі часові кроки просування стану відрізняються.

1.2.2 Підходи до синхронізації в паралельних алгоритмах імітації

Проблема синхронізації між частинами моделі в асинхронному методі просторової декомпозиції визначає ефективність отриманого в результаті паралельного алгоритму. При занадто частій синхронізації алгоритм буде неефективним, а якщо виконувати синхронізацію рідко, результат буде не точним, або зростуть затрати пам’яті. У роботі [3] наведено два способи синхронізації:

– консервативний: виконання кожної частини моделі блокується, поки не можна гарантувати, що з сусідніх частин ніколи не надійде подія с часовою позначкою меншу за визначену. Це досягається шляхом повідомлень, які частини моделі посилають одна одній під час виникнення межових подій. Оскільки повідомлення надсилаються тільки під час виникнення внутрішніх

межових подій, у наступних повідомлень позначка часу завжди буде більшою за попередні, тому що час у моделях просувається тільки в одну сторону. На жаль, такий підхід призводить до дедлоку, коли частини моделей утворюють цикл і кожна чекає на повідомлення від попередньої, щоб визначити інтервал часу, в якому гарантовано не буде зовнішніх подій. Для вирішення цієї проблеми частини моделі повинні додатково надсилати повідомлення з часовою відміткою їх найближчої внутрішньої події. Оскільки час найближчої події не більший за межову подію, сусідня частина моделі може безпечно імітуватися в цьому інтервалі. Однак, повідомлення з часом найближчої внутрішньої події приводять до неефективності, тому що стрімко зростає частота синхронізації між частинами моделі;

– оптимістичний: виконання кожної частини моделі не блокується, але при надходженні повідомлення від сусідньої частини, якщо часова позначка в ньому менша за поточний час в частині, виконується відкат. За час роботи частини моделі в неправильному порядку могли відбутися два типи подій: зміна внутрішнього стану та надсилання повідомлення сусіднім частинам моделі. Для відкату стану частини моделі необхідно зберігати знімки її станів у певний час. Щоб розв’язати проблему з повідомленнями потрібно надіслати так зване анти-повідомлення, яке знищить початкове повідомлення, якщо воно ще не було опрацьовано, або запустить відкат у цій сусідній частині. Рекурсивно ця процедура повністю прибере ефект від помилки. Цей спосіб має дві центральні проблеми: відкат може займати дуже багато часу, якщо якась частина моделі встигла дуже далеко просунути свій локальний час, та на зберігання знімків моделі витрачається дуже багато пам’яті.

1.2.3 Паралельний алгоритм імітації Петрі-об’єктних моделей

Алгоритм імітації Петрі-об’єктних моделей розглянуто у роботі [4]. В цьому алгоритмі використовується підхід просторової декомпозиції, а синхронізація виконується консервативним способом. Петрі-об’єктна модель

розділяється на Петрі-об’єкти, які самі просувають свій локальний час. У наведеного алгоритму є декілька обмежень, що не дозволяють йому бути універсальним:

– кожен Петрі-об’єкт може мати максимум один сусідній об’єкт, який надсилає йому повідомлення;

– кожен Петрі-об’єкт може мати максимум один сусідній об’єкт, якому він надсилає повідомлення;

– Петрі-об’єкти в моделі не мають утворювати цикл.

Для подолання перших двох обмежень необхідно замість одного буферу зовнішніх подій мати на Петрі-об’єкті декілька таких буферів, по одному на кожне з’єднання з іншим Петрі-об’єктом, який буде надсилати повідомлення. При такому розширенні алгоритму потрібно змінити правила визначення межі безпечного інтервалу моделювання та тимчасове зупинення роботи при перевищенні ліміту зовнішніх подій.

Межа безпечного інтервалу буде визначатися за наступними правилами:

а) серед усіх буферів зовнішніх подій знайти першу подію з буфера з мінімальною часовою позначкою;

б) якщо подія знайдена і її часова позначка менша за час моделювання, то межа моделювання встановлюється в цю часову позначку інакше в час моделювання.

Ліміт зовнішніх подій має встановлюватися на мінімальний розмір з буферів зовнішніх подій, у який Петрі-об’єкт надсилає повідомлення.

Щоб подолати обмеження з утворенням циклу потрібно перед обчисленням Петрі-об’єктної моделі провести її перебудову. Оскільки Петрі-об’єктна модель утворює орієнтований граф, задачу можна звести до пошуку компонентів сильної зв’язності орієнтованого графа. За допомогою алгоритму Косараджу можна знайти усі компоненти сильної зв’язності, а потім об’єднати Петрі-об’єкти, що утворюють ці компоненти у більші Петрі-об’єкти.

Окремо необхідно розглянути проблему Петрі-об’єктів об’єднаних через спільні позиції. Між двома Петрі-об’єктами, які поєднані через спільну позицію відбуваються два види взаємодії: вихід маркерів в спільну позицію та забирання маркерів зі спільної позиції. Повідомлення про ці взаємодії об’єкти також мають передавати один одному в буфер повідомлень. Однак, при визначенні безпечного інтервалу роботи об’єкти будуть блокуватися один на одному саме через те, що вони повинні передавати повідомлення один одному, а це утворює цикл. Тому Петрі-об’єкти, які поєднані через спільну позицію мають сприйматися як такі, що утворюють цикл один з одним, та розв’язуватися так само, як й інші цикли.

1.3 Аналіз успішних ІТ-проектів

Ринок програмних продуктів для імітації роботи систем, хоч і не сильно наповнений, та доволі довгий час існує. Найстаріші додатки з великою кількістю оновлень набули різноманітного функціоналу та продовжують впроваджувати нові підходи.

До найпопулярніших програм для імітації можна віднести:

Arena Simulation

Найбільш поширений продукт у сфері імітації. Загалом можна виділити наступні переваги:

- різноманіття блоків, з яких конструюється модель;
- проста ієрархічна модель;
- добре реалізована анімація роботи моделі;
- повна статистика, та генерація звітів.

Серед недоліків важливо виділити неможливість створювати Петрі-об’єктну модель імітації, що позбавляє користувача потужного інструмента.

CPNTools

Найпопулярнішим продуктом, який підтримує мережі Петрі є CPNTools. Серед його переваг можна виділити основні:

– підтримка кольорових мереж Петрі, які значно зменшують вигляд моделі;

– підтримка ієрархічних мереж Петрі;

– підтримка стохастичних мереж Петрі;

– можливість задати умову завершення імітації, а не тільки ліміт часу;

– можливість тимчасово зупиняти імітацію та змінювати стан моделі.

Все ж імітація мереж Петрі в CPNTools виконується послідовно, що займає занадто багато часу на великих моделях.

HiPS

Hierarchical Petri net Simulator не дуже відомий програмний продукт, яле його варто розглянути, адже у ньому реалізовані:

– підтримка стохастичних мереж Петрі;

– паралельна імітація;

– знаходження T- і S-інваріантів;

– аналіз мережі Петрі на досяжність та дедлоки.

Варто відзначити, що хоча у HiPS і реалізована паралельна імітація, вона підтримується лише частково.

1.4 Аналіз вимог до програмного забезпечення

Вимоги до програмного продукту ставляться з оглядом на три речі: що програмний продукт має робити, кому потрібен цей функціонал, та навіщо він потрібен. Таким чином для визначення вимог по програмного забезпечення потрібно виділити ролі користувачів, функціональні вимоги, які відносяться до цих ролей та пояснення до цих вимог. Окремо ставляться нефункціональні вимоги, в яких описуються характеристики результуючого продукту, та певні обмеження на те, як має бути реалізований функціонал.

Оскільки результуючим продуктом є програмна бібліотека, можна виділити лише одну роль користувача — клієнт-програміст (далі Програміст).

Хоча роль одна, вимоги до програмної бібліотеки мають бути, адже такий продукт може використовуватися як основна бібліотека в різноманітних сервісах для імітації Петрі-об’єктних моделей, або простих Петрі мереж, наприклад:

– нативний графічний додаток;

– веб-додаток;

– програма для дослідження, яка розробляється, щоб якнайшвидше провести експеримент.

Отже, Програміст повинен мати змогу отримувати максимальну кількість інформації з імітації системи та детальну статистику результату обчислення моделі.

1.4.1 Розроблення функціональних вимог

Для повноти функціональних умов необхідно розглянути усі сценарії роботи з програмною бібліотекою. Сценарії можна представити у вигляді варіантів використання. Передбачені варіанти використання представлені у таблицях 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12.

Таблиця 1.1 – Варіант використання номер UC001

Номер	UC001
Назва	Імітація Петрі-об’єктної моделі з багатоканальними переходами
Опис	Програміст має можливість запустити імітацію Петрі-об’єктної моделі з багатоканальними переходами
Учасники	Програміст
Постумови	Програміст отримує протокол подій та результати виконання імітації

Продовження таблиці 1.1

Основний сценарій	Програміст створює валідну Перті-об'єктну модель аналогічну зображеній на рисунку 1.1, у якій усі переходи багатоканальні; Програміст передає Петрі-об'єктну модель на послідовне чи паралельне виконання; Бібліотека коректно імітує цю модель та повертає справедливі результати
Розширення сценаріїв	

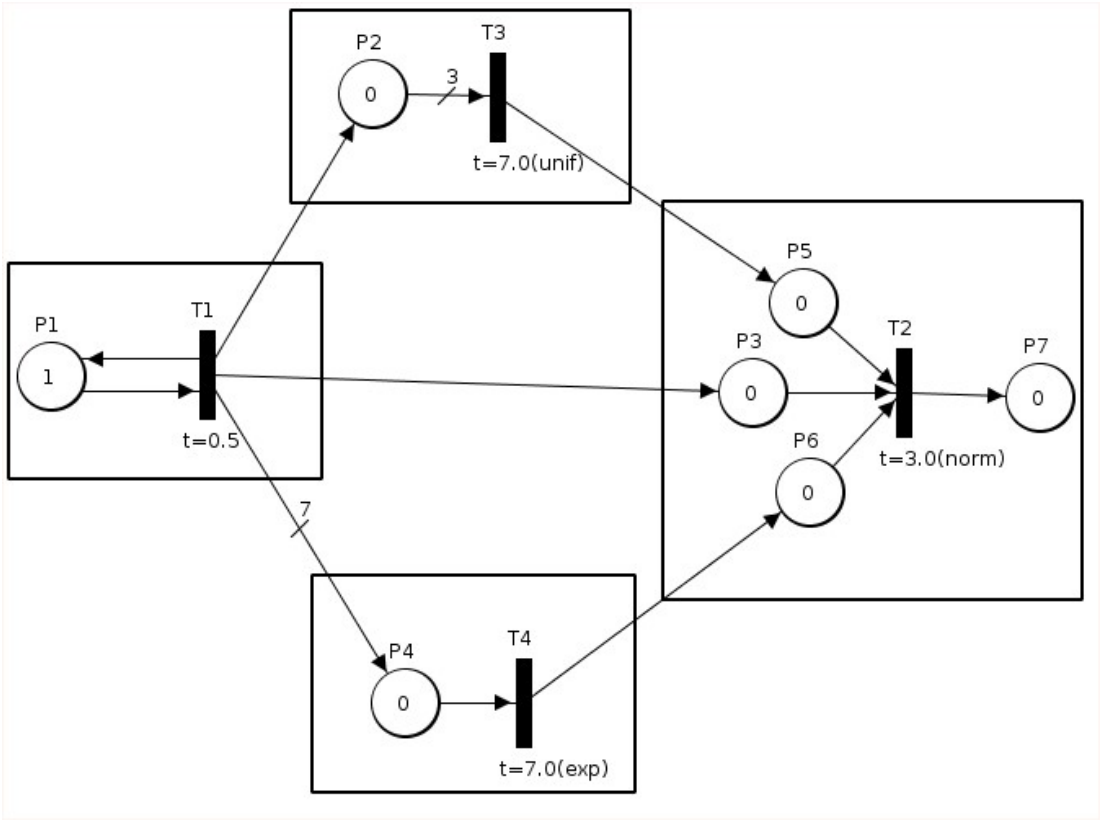


Рисунок 1.1 – Приклад Петрі-об'єктної моделі з багатоканальними переходами, багатократними дугами, константними затримками в переходах, з випадковими затримками в переходах розподіленими рівномірно на відрізку, за експоненціальним та за нормальним розподілом

Таблиця 1.2 – Варіант використання номер UC002

Номер	UC002
Назва	Імітація Петрі-об’єктної моделі з багатократними дугами
Опис	Програміст має можливість запустити імітацію Петрі-об’єктної моделі з багатократними дугами
Учасники	Програміст
Постумови	Програміст отримує протокол подій та результати виконання імітації
Основний сценарій	Програміст створює валідну Перті-об’єктну модель аналогічну зображеній на рисунку 1.1, у якій є багатократні дуги; Програміст передає Петрі-об’єктну модель на послідовне чи паралельне виконання; Бібліотека коректно імітує цю модель та повертає справедливі результати
Розширення сценаріїв	

Таблиця 1.3 – Варіант використання номер UC003

Номер	UC003
Назва	Імітація Петрі-об’єктної моделі з константними затримками в переходах
Опис	Програміст має можливість запустити імітацію Петрі-об’єктної моделі з константними затримками в переходах
Учасники	Програміст
Постумови	Програміст отримує протокол подій та результати виконання імітації

Продовження таблиці 1.3

Основний сценарій	Програміст створює валідну Перті-об’єктну модель аналогічну зображеній на рисунку 1.1, у якій є константні затримки в переходах; Програміст передає Петрі-об’єктну модель на послідовне чи паралельне виконання; Бібліотека коректно імітує цю модель та повертає справедливі результати
Розширення сценаріїв	У випадку, коли константна затримка менша за нуль, Програміст повинен отримати виключення з помилкою при створенні моделі

Таблиця 1.4 – Варіант використання номер UC004

Номер	UC004
Назва	Імітація Петрі-об’єктної моделі з затримками в переходах, вказаних у вигляді рівномірно розподіленої випадкової величини
Опис	Програміст має можливість запустити імітацію Петрі-об’єктної моделі з затримками в переходах, вказаних у вигляді рівномірно розподіленої випадкової величини
Учасники	Програміст
Постумови	Програміст отримує протокол подій та результати виконання імітації

Продовження таблиці 1.4

Основний сценарій	Програміст створює валідну Перті-об’єктну модель аналогічну зображеній на рисунку 1.1, у якій є затримки в переходах, вказані у вигляді рівномірно розподіленої випадкової величини; Програміст передає Петрі-об’єктну модель на послідовне чи паралельне виконання; Бібліотека коректно імітує цю модель та повертає справедливі результати
Розширення сценаріїв	У випадку, коли мінімальна межа інтервалу, на якому розподіляються випадкові величини, менша за нуль, або більша за максимальну межу, Програміст повинен отримати виключення з помилкою при створенні моделі

Таблиця 1.5 – Варіант використання номер UC005

Номер	UC005
Назва	Імітація Петрі-об’єктної моделі з затримками в переходах, вказаних у вигляді випадкової величини розподіленої за експоненціальним законом
Опис	Програміст має можливість запустити імітацію Петрі-об’єктної моделі з затримками в переходах, вказаних у вигляді випадкової величини розподіленої за експоненціальним законом
Учасники	Програміст
Постумови	Програміст отримує протокол подій та результати виконання імітації

Продовження таблиці 1.5

Основний сценарій	Програміст створює валідну Перті-об’єктну модель аналогічну зображеній на рисунку 1.1, у якій є затримки в переходах, вказані у вигляді випадкової величини розподіленої за експоненціальним законом; Програміст передає Петрі-об’єктну модель на послідовне чи паралельне виконання; Бібліотека коректно імітує цю модель та повертає справедливі результати
Розширення сценаріїв	У випадку, коли значення середньої затримки експоненціального закону менше або дорівнює нулю, Програміст повинен отримати виключення з помилкою при створенні моделі

Таблиця 1.6 – Варіант використання номер UC006

Номер	UC006
Назва	Імітація Петрі-об’єктної моделі з затримками в переходах, вказаних у вигляді випадкової величини розподіленої за нормальним законом
Опис	Програміст має можливість запустити імітацію Петрі-об’єктної моделі з затримками в переходах, вказаних у вигляді випадкової величини розподіленої за нормальним законом
Учасники	Програміст
Постумови	Програміст отримує протокол подій та результати виконання імітації

Продовження таблиці 1.6

Основний сценарій	Програміст створює валідну Перті-об’єктну модель аналогічну зображеній на рисунку 1.1, у якій є затримки в переходах, вказані у вигляді випадкової величини розподіленої за нормальним законом; Програміст передає Петрі-об’єктну модель на послідовне чи паралельне виконання; Бібліотека коректно імітує цю модель та повертає справедливі результати
Розширення сценаріїв	У випадку, коли значення відхилення затримки нормального закону менше або дорівнює нулю, Програміст повинен отримати виключення з помилкою при створенні моделі. Якщо при генерації випадкових чисел отримана затримка, яка менша за нуль, спроба генерації має повторитися, поки отримана затримка не буде більшою або рівною нулю

Таблиця 1.7 – Варіант використання номер UC007

Номер	UC007
Назва	Імітація Петрі-об’єктної моделі з конфліктними переходами
Опис	Програміст має можливість запустити імітацію Петрі-об’єктної моделі з конфліктними переходами, які вирішуються за пріоритетом, або за вказаною імовірністю
Учасники	Програміст
Постумови	Програміст отримує протокол подій та результати виконання імітації

Продовження таблиці 1.7

Основний сценарій	Програміст створює валідну Перті-об'єктну модель аналогічну зображеній на рисунку 1.2, у якій є конфліктні переходи і вказані пріоритети або ймовірності цих переходів; Програміст передає Петрі-об'єктну модель на послідовне чи паралельне виконання; Бібліотека коректно імітує цю модель та повертає справедливі результати
Розширення сценаріїв	У випадку, коли імовірність переходу не вказана, вона має бути рівною одиниці; Якщо вказані і ймовірність, і пріоритет, конфлікт має спочатку вирішуватися за пріоритетом, а потім за ймовірністю серед переходів, що залишилися; Якщо сума ймовірностей переходів при вирішенні конфлікту не дорівнює одиниці, імовірності мають бути пропорційно змінені так, щоб їх сума була рівною одиниці

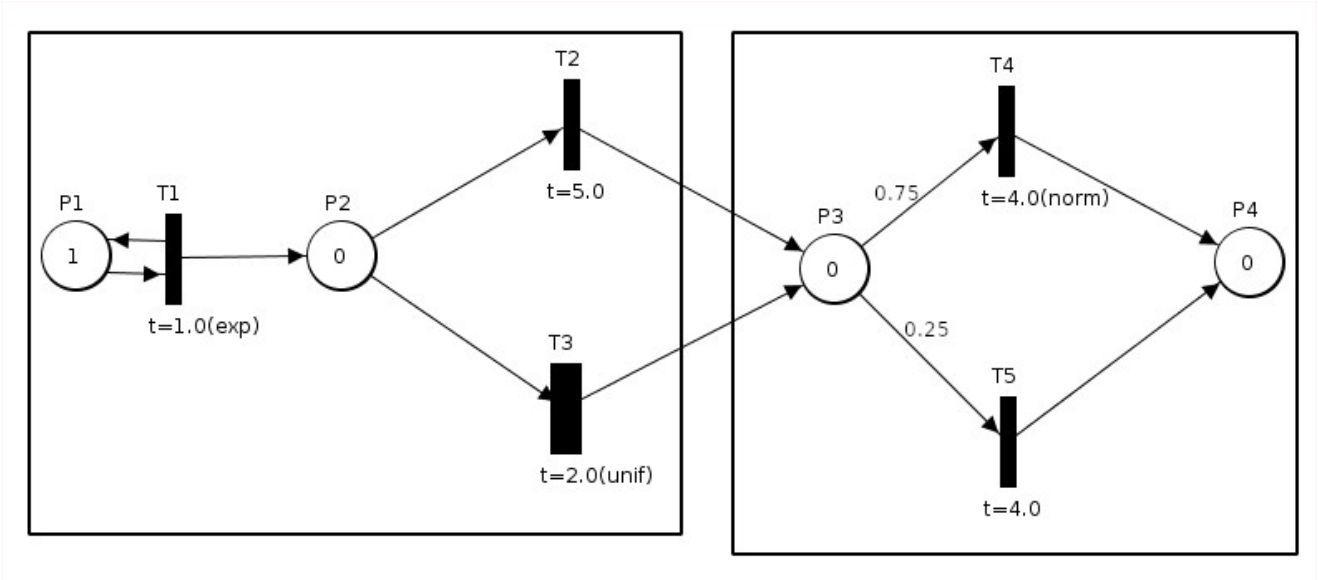


Рисунок 1.2 – Приклад Петрі-об'єктної моделі з конфліктними переходами

Таблиця 1.8 – Варіант використання номер UC008

Номер	UC008
Назва	Імітація Петрі-об’єктної моделі з інформаційними вхідними дугами
Опис	Програміст має можливість запустити імітацію Петрі-об’єктної моделі з інформаційними вхідними дугами
Учасники	Програміст
Постумови	Програміст отримує протокол подій та результати виконання імітації
Основний сценарій	Програміст створює валідну Перті-об’єктну модель аналогічну зображеній на рисунку 1.3, у якій є інформаційні вхідні дуги; Програміст передає Петрі-об’єктну модель на послідовне чи паралельне виконання; Бібліотека коректно імітує цю модель та повертає справедливі результати
Розширення сценаріїв	

Рисунок 1.3 – Приклад Петрі-об’єктної моделі з інформаційними дугами

Таблиця 1.9 – Варіант використання номер UC009

Номер	UC009
Назва	Валідація цілісності Петрі-об’єктної моделі
Опис	Програміст має отримати помилку при намаганні імітувати Петрі-об’єктну моделі, у якій є переходи без вхідних або вихідних дуг
Учасники	Програміст
Постумови	Програміст отримає виключення з помилкою валідації
Основний сценарій	Програміст створює Петрі-об’єктну модель аналогічну зображеній на рисунку 1.4, у якій є переходи без вхідних або без вихідних дуг; Програміст передає Петрі-об’єктну модель на послідовне чи паралельне виконання; Бібліотека повертає виключення з помилкою валідації

Продовження таблиці 1.9

Розширення сценаріїв	

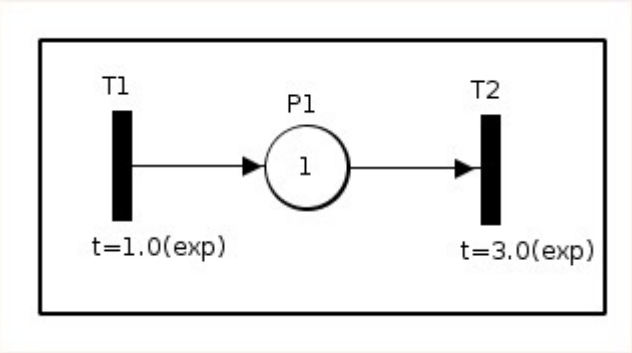


Рисунок 1.4 – Приклад не цілісної Петрі-об’єктної моделі

Таблиця 1.10 – Варіант використання номер UC010

Номер	UC010
Назва	Валідація вхідних інформаційних дуг Перті-об’єктної моделі
Опис	Програміст має отримати помилку при намаганні імітувати Петрі-об’єктну модель, у якій є перехід з інформаційною вхідною дугою, у якого немає неінформаційної вхідної дуги
Учасники	Програміст
Постумови	Програміст отримає виключення з помилкою валідації
Основний сценарій	Програміст створює Петрі-об’єктну модель аналогічну зображеній на рисунку 1.5, у якій є перехід з інформаційною вхідною дугою, у якого немає неінформаційної вхідної дуги; Програміст передає Петрі-об’єктну модель на послідовне чи паралельне виконання; Бібліотека повертає виключення з помилкою валідації
Розширення сценаріїв	

Рисунок 1.5 – Приклад Петрі-об’єктної моделі з переходом, у якого є інформаційна вхідна дуга та немає неінформаційних вхідних дуг

Таблиця 1.11 – Варіант використання номер UC011

Номер	UC011
Назва	Паралельна імітація Петрі-об’єктної моделі, у якій об’єкти утворюють цикл
Опис	Програміст має можливість запустити імітацію Петрі-об’єктної моделі, у якій об’єкти утворюють цикл
Учасники	Програміст
Постумови	Програміст отримує протокол подій та результати виконання імітації

Продовження таблиці 1.11

Основний сценарій	Програміст створює Петрі-об’єктну модель аналогічну зображеній на рисунку 1.6, у якій Петрі-об’єкти утворюють цикл; Програміст передає Петрі-об’єктну модель на паралельне виконання; Бібліотека знаходить усі компоненти сильної зв’язності у Петрі-об’єктній моделі, та створює з них більші Петрі-об’єкти, так, щоб результуюча Петрі-об’єктна модель не мала циклів; Бібліотека коректно імітує цю модель та повертає справедливі результати
Розширення сценаріїв	

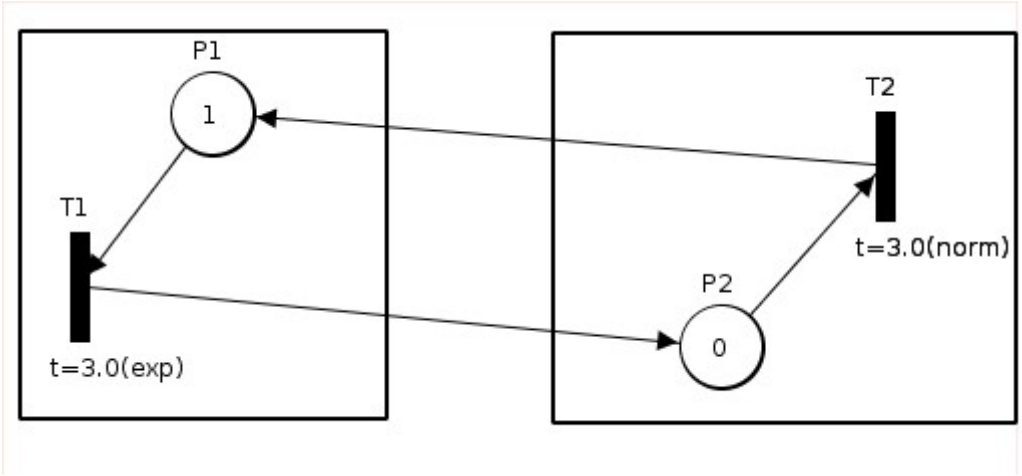


Рисунок 1.6 – Приклад Петрі-об’єктної моделі, у якій об’єкти утворюють цикл

Таблиця 1.12 – Варіант використання номер UC012

Номер	UC012
Назва	Паралельна імітація Петрі-об’єктної моделі, у якій об’єкти об’єднані через спільну позицію

Продовження таблиці 1.12

Опис	Програміст має можливість запустити імітацію Петрі-об'єктної моделі, у якій об'єкти об'єднані через спільну позицію
Учасники	Програміст
Постумови	Програміст отримує протокол подій та результати виконання імітації
Основний сценарій	Програміст створює Петрі-об'єктну модель аналогічну зображеній на рисунку 1.7, у якій Петрі-об'єкти об'єднані через спільну ініціалізацію подій; Програміст передає Петрі-об'єктну модель на паралельне виконання; Бібліотека створює з кожних об'єктів, об'єднаних через спільну позицію, більший об'єкт так, щоб результуюча Петрі-об'єктна модель не мала об'єктів, об'єднаних через спільну позицію; Бібліотека коректно імітує цю модель та повертає справедливі результати
Розширення сценаріїв	

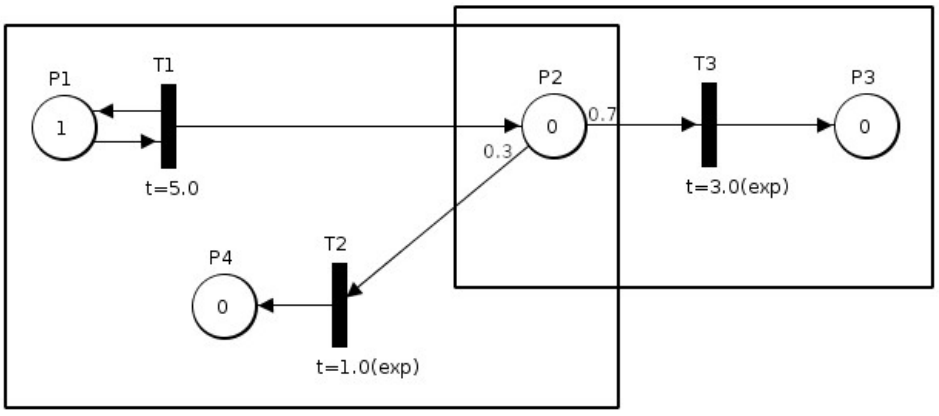


Рисунок 1.7 – Приклад Петрі-об'єктної моделі, у якій об'єкти об'єднані через спільну ініціалізацію подій

З цих варіантів використання можна виділити функціональні вимоги, наведені в таблицях 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, 1.19, 1.20, 1.21, 1.22, 1.23, 1.24, 1.25, 1.26, 1.27, 1.28:

Таблиця 1.13 – Функціональна вимога номер REQ001

Номер	REQ001
Назва	Багатоканальні переходи
Опис	Бібліотека має імітувати Петрі-об’єктні моделі з багатоканальними переходами

Таблиця 1.14 – Функціональна вимога номер REQ002

Номер	REQ002
Назва	Багатократні дуги
Опис	Бібліотека має імітувати Петрі-об’єктні моделі з багатократними дугами

Таблиця 1.15 – Функціональна вимога номер REQ003

Номер	REQ003
Назва	Константні затримки в переходах
Опис	Бібліотека має імітувати Петрі-об’єктні моделі з константними затримками в переходах

Таблиця 1.16 – Функціональна вимога номер REQ004

Номер	REQ004
Назва	Випадкові затримки в переходах рівномірно розподілені на інтервалі
Опис	Бібліотека має імітувати Петрі-об’єктні моделі з випадковими затримками в переходах рівномірно розподіленими на інтервалі

Таблиця 1.17 – Функціональна вимога номер REQ005

Номер	REQ005
Назва	Випадкові затримки в переходах розподілені за експоненціальним законом
Опис	Бібліотека має імітувати Петрі-об’єктні моделі з випадковими затримками в переходах розподіленими за експоненціальним законом

Таблиця 1.18 – Функціональна вимога номер REQ006

Номер	REQ006
Назва	Випадкові затримки в переходах розподілені за нормальним законом
Опис	Бібліотека має імітувати Петрі-об’єктні моделі з випадковими затримками в переходах розподіленими за нормальним законом

Таблиця 1.19 – Функціональна вимога номер REQ007

Номер	REQ007
Назва	Конфліктні переходи
Опис	Бібліотека має імітувати Петрі-об’єктні моделі з конфліктними переходами, розв’язання конфліктів між якими виконується за пріоритетом та ймовірністю

Таблиця 1.20 – Функціональна вимога номер REQ008

Номер	REQ008
Назва	Інформаційні вхідні дуги
Опис	Бібліотека має імітувати Петрі-об’єктні моделі з інформаційними вхідними дугами

Таблиця 1.21 – Функціональна вимога номер REQ009

Номер	REQ009
Назва	Валідація коректності створеної Петрі-об’єктної моделі
Опис	Бібліотека має валідувати коректності створеної Петрі-об’єктної моделі

Таблиця 1.22 – Функціональна вимога номер REQ010

Номер	REQ010
Назва	Розв’язання циклів Петрі-об’єктів при паралельній імітації
Опис	Бібліотека має переробляти кожен цикл Перті-об’єктів в один Петрі-об’єкт перед для паралельної імітації

Таблиця 1.23 – Функціональна вимога номер REQ011

Номер	REQ011
Назва	Протокол подій

Продовження таблиці 1.23

Опис	Бібліотека має мати інтерфейс для отримання протоколу подій, у якому передається інформація про всі події, які виникають, зміна стану елементів, на яких виникають події та час події

Таблиця 1.24 – Функціональна вимога номер REQ012

Номер	REQ012
Назва	Статистика
Опис	Після імітації бібліотека має повертати статистику стану елементів моделі, які Програміст позначив як ті, що вимагають статистики

Таблиця 1.25 – Функціональна вимога номер REQ013

Номер	REQ013
Назва	Коректна імітація
Опис	Бібліотека має імітувати Петрі-об’єктну модель так, щоб результати були відрізнялися від аналітичних не більше ніж на 5%

Таблиця 1.26 – Функціональна вимога номер REQ015

Номер	REQ014
Назва	Паралельна імітація Петрі-об’єктної моделі з декількома попередніми та наступними об’єктами у Петрі-об’єкта
Опис	Бібліотека має імітувати Петрі-об’єктну модель, у якої є Петрі-об’єкти, які отримують повідомлення від декількох інших Петрі-об’єктів, та які передають повідомлення декільком Петрі-об’єктам

Таблиця 1.27 – Функціональна вимога номер REQ015

Номер	REQ015
Назва	Імітація однієї Петрі-об’єктної моделі і послідовно, і паралельно без перебудови моделі
Опис	Якщо Петрі-об’єкти в Петрі-об’єктній моделі зв’язані через зовнішні позиції, бібліотека має імітувати цю модель і послідовно, і паралельно без потреби Програмісту перебудовувати модель

Таблиця 1.28 – Функціональна вимога номер REQ016

Номер	REQ016
Назва	Паралельна імітація Петрі-об’єктної моделі зв’язаної через спільну ініціалізацію подій при паралельній імітації
Опис	Бібліотека має імітувати Петрі-об’єктну модель, у якій є Петрі-об’єкти, які зв’язані через спільну позицію

Покриття вимогами до бібліотеки паралельних обчислень Петрі-об’єктних моделей сценаріїв її використання зображено на рисунку 1.8.

	REQ001 Багатоканальні переходи	REQ002 Багатократні дуги	REQ003 Константні затримки в переходах	REQ004 Випадкові затримки в переходах рівномірно розподілені на інтервалі	REQ005 Випадкові затримки в переходах розподілені за експоненціальним законом	REQ006 Випадкові затримки в переходах розподілені за нормальним законом	REQ007 Конфліктні переходи	REQ008 Інформаційні вхідні дуги	REQ009 Валідація коректності створенної Петрі-об'єктної моделі	REQ010 Розв'язання циклів Петрі-об'єктів при паралельній імітації	REQ011 Протокол подій	REQ012 Статистика	REQ013 Коректна імітація	REQ014 Паралельна імітація Петрі-об'єктної моделі з декількома попередніми та наступними об'єктами у Петрі-об'єкта	REQ015 Імітація однієї Петрі-об'єктної моделі і послідовно, і паралельно без перебудови моделі	REQ016 Паралельна імітація Петрі-об'єктної моделі зв'язаної через спільну ініціалізацію подій при паралельній імітації
UC001 Імітація Петрі-об'єктної моделі з багатоканальними переходами	■	■	■	■	■	■					■	■	■	■	■	
UC002 Імітація Петрі-об'єктної моделі з багатократними дугами	■	■	■	■	■	■					■	■	■	■	■	
UC003 Імітація Петрі-об'єктної моделі з константними затримками в переходах	■	■	■	■	■	■					■	■	■	■	■	
UC004 Імітація Петрі-об'єктнох моделі з затримками в переходах, вказаних у вигляді рівномірно розподіленої випадкової величини	■	■	■	■	■	■					■	■	■	■	■	
UC005 Імітація Петрі-об'єктної моделі з затримками в переходах, вказаних у вигляді випадкової величини розподіленої за експоненціальним законом	■	■	■	■	■	■					■	■	■	■	■	
UC006 Імітація Петрі-об'єктної моделі з затримками в переходах, вказаних у вигляді випадкової величини розподіленої за нормальним законом	■	■	■	■	■	■					■	■	■	■	■	
UC007 Імітація Петрі-об'єктної моделі з конфліктними переходами	■		■	■	■	■	■				■	■	■	■	■	
UC008 Імітація Петрі-об'єктної моделі з інформаційними вхідними дугами	■		■		■			■			■	■	■		■	
UC009 Валідація цілісності Петрі-об'єктної моделі									■							
UC010 Валідація вхідних інформаційних дуг Перті-об'єктної моделі									■							
UC011 Паралельна імітація Петрі-об'єктної моделі, у якій об'єкти утворюють цикл	■				■	■				■	■	■	■			
UC012 Паралельна імітація Петрі-об'єктної моделі, у якій об'єкти об'єднанч через спільну позицію	■		■		■		■				■	■	■		■	■

Рисунок 1.8 — Матриця покриття вимогами сценаріїв використання

1.4.2 Розроблення нефункціональних вимог

Бібліотеці паралельних обчислень Петрі-об’єктних моделей виставляються наступні нефункціональні умови:

– має бути виокремлений модуль з об’єктною моделлю, яку має бути легко створити, наприклад, з JSON, та навпаки перетворити у якийсь формат. Стан цієї моделі не має змінюватися під час імітації, для цього необхідно виділити окремий модуль;

– паралелізм має пришвидшити імітацію хоча б у півтора рази;

– паралелізм має бути реалізованим на пулі потоків, потоки у якому можуть перемикатися на імітацію іншого вільного Петрі-об’єкта, якщо імітація його Петрі-об’єкта заблокована;

– знаходження компонентів сильної зв’язності Петрі-об’єкті при паралельній імітації має бути реалізовано алгоритмом Косараджу;

– бібліотека має бути написаною мовою Java для простої інтеграцію з уже готовими програмними продуктами;

– одним з варіантів тестування має бути Петрі мережа зображена на рисунку 1.9. Після імітації отримані середні значення мають бути близькими до аналітичних, а саме: Queue1 — 1,786, Queue2 — 0,003, Queue3 — 0,004, Queue4 — 0,00001, Limit 1 — 0,268, Limit 2 — 0,946, Limit 3 — 0,938, Limit 4 — 1.964.

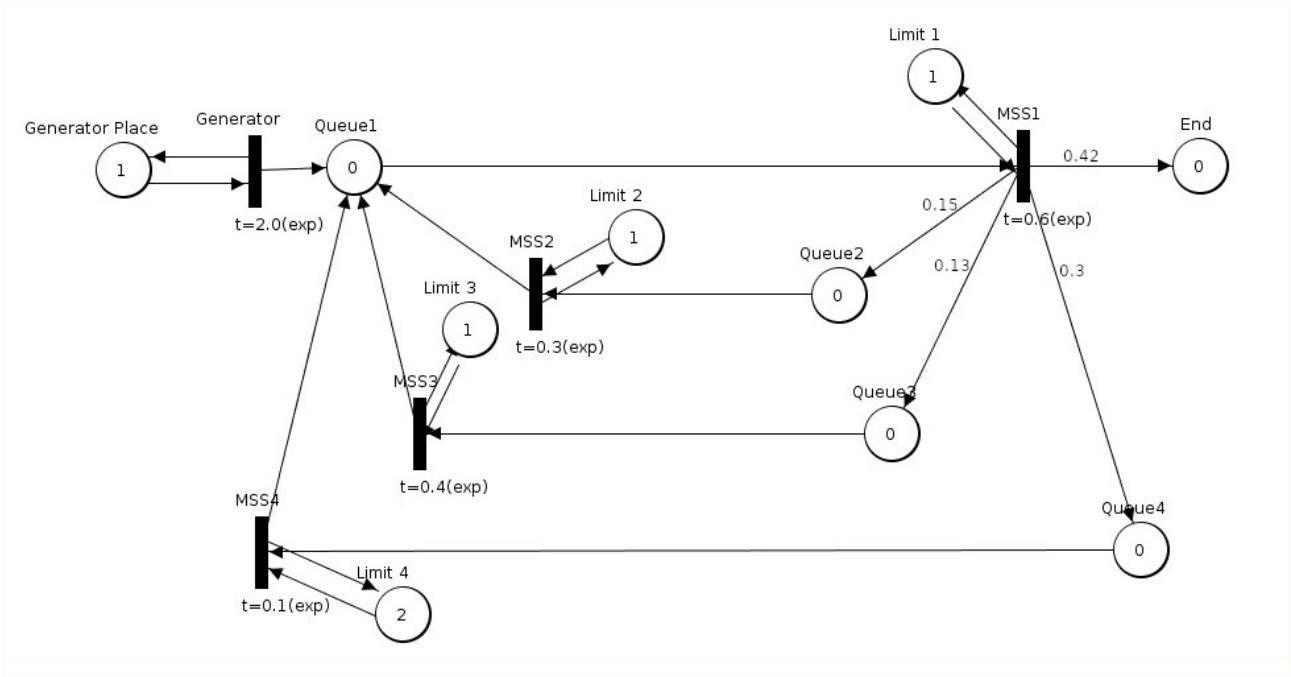


Рисунок 1.9 — Варіант мережі Петрі для тестування бібліотеки

1.4.3 Постановка комплексу завдань модулю

Бібліотека паралельного обчислювання Петрі-об’єктних моделей має зменшити час імітацій, які використовуються при розробці, аналізі та оптимізації різноманітних продуктів та бізнес-процесів. Отже, мета цієї розробки — підвищити швидкодію алгоритму імітації Петрі-об’єктної моделі та представити це у вигляді лаконічної гнучкої бібліотеки.

Для того, щоб досягти поставленої мети в бібліотеці мають бути реалізовані наступні можливості:

– коректна послідовна та паралельна імітація Петрі-об’єктної моделі;

– повноцінна опціональна статистика стану елементів Петрі-об’єктної моделі;

– протокол подій в межах Петрі-об’єктної моделі для послідовної імітації та в межах кожного Петрі-об’єкта для паралельної імітація, який передає повноцінну інформацію про кожну подію імітації.

Бібліотека повинна мати добре розділені компоненти та бути реалізованою мовою Java.

1.5 Висновки по розділу

У сучасному світі моделювання як інструмент налагодження роботи та процесів стає невід'ємною частиною багатьох сфер діяльності: логістика, виробництво, розробка транспортних шляхів, обслуговування клієнтів та аналіз клієнтського досвіду, громадський транспорт, комп'ютерні мережі та програмні додатки – лише невелика частина з можливих шляхів використання моделювання.

Наразі існує ряд рішень, що можуть допомогти пришвидшити та зробити більш ефективним процес моделювання. Деякі з них пропонують моделі не у вигляді Петрі-об'єктної бібліотеки, що значно впливає на гнучкість. Інші ж, використовуючи послідовну реалізацію обчислень поступаються по швидкості.

Бібліотека паралельних обчислень Петрі-об'єктних моделей якнайкраще проявляє себе під час розробки, оптимізації, контролю та аналізу модельованих систем. До її відмінних особливостей можна віднести: імітацію Петрі-об'єктних моделей з багатоканальними переходами, багатократними дугами та константними затримками у переходах, імітацію Петрі-об'єктних моделей з випадковими затримками в переходах рівномірно розподіленими на інтервалі, розподілення випадкових затримок за експоненціальним та за нормальним законами, розв'язання конфліктів між переходами за пріоритетом та ймовірністю, валідація коректності створеної Петрі-об'єктної моделі тощо. Також бібліотека паралельних обчислень Петрі-об'єктних моделей здатна на надання протоколу, де зберігається вся інформація про всі виникаючі події, стан та зміну станів елементів, на яких виникають події та час події. Для зручного користування передбачена функція повернення статистики, яку користувач відмітив для себе, як важливу.

Тож, бібліотека паралельних обчислень Петрі-об'єктних моделей здатна значно облегшити процеси моделювання у багатьох сферах, зменшити час проведення імітацій та підвищити ефективність моделювання як інструменту для роботи у сферах виробництва.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Для створення належної архітектури програмної бібліотеки необхідно формалізувати сценарії використання та функціональні вимоги у вигляді бізнес-процесів, представлених, наприклад, BPMN діаграмами. Для програмного продукту, що розробляється можна виділити чотири бізнес-процеси:

– створення Петрі-об’єкта;

– створення Петрі-об’єктної моделі;

– послідовна імітація Петрі-об’єктної моделі;

– паралельна імітація Петрі-об’єктної моделі.

Схема бізнес-процесу створення Петрі-об’єкта зображена на рисунку 2.1.

Опис бізнес-процесу створення Петрі-об’єкта:

– програміст передає бібліотеці ідентифікатори та імена позиції. На цьому етапі у нього також є можливість задати початкове маркування позицій. Маркування позиції, для якої початкове маркування не вказано вважається нульовим;

– бібліотека повертає Програмісту об’єктну модель позицій;

– програміст передає бібліотеці ідентифікатори та імена переходів. Опціонально він може задати пріоритет переходів (за замовчуванням нуль), імовірність переходів (за замовчуванням одиниця), закон розподілу затримки в переходах (за замовчуванням константний з нульовою затримкою);

– бібліотека повертає об’єктну модель переходів;

– програміст передає бібліотеці об’єктні моделі переходів і позицій разом з інформацією про зв’язки між ними;

– бібліотека повертає об’єктну модель Петрі-об’єкта.

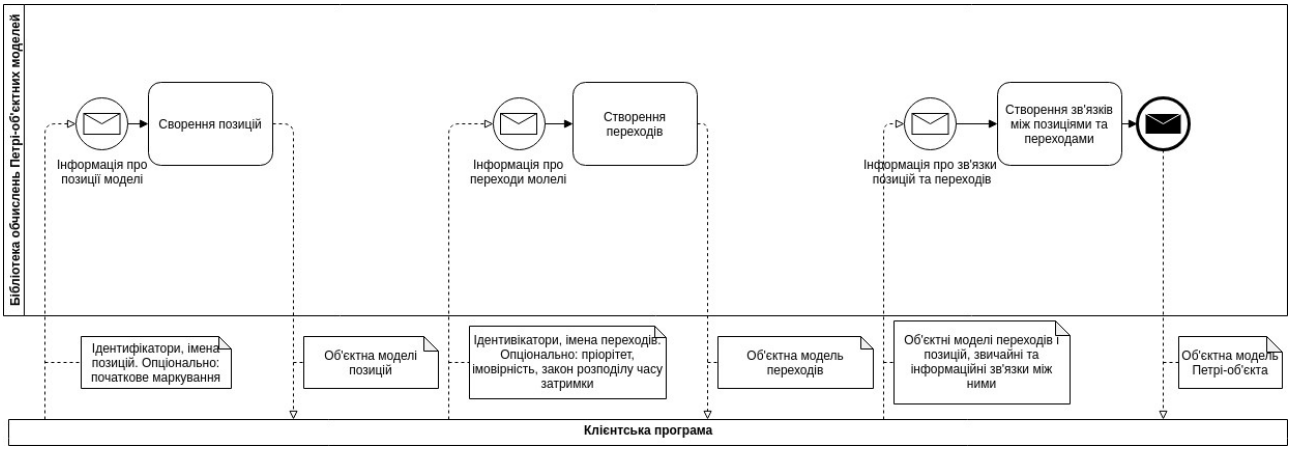


Рисунок 2.1 – Схема бізнес-процесу створення Петрі-об’єкта

Схема бізнес-процесу створення Петрі-об’єктної моделі зображена на рисунку 2.2.

Опис бізнес-процесу створення Петрі-об’єктної моделі:

– програміст передає бібліотеці об’єктні моделі Петрі-об’єктів та інформацію про зв’язки між ними;

– бібліотека повертає програмісту об’єктну модель Петрі-об’єктної моделі.

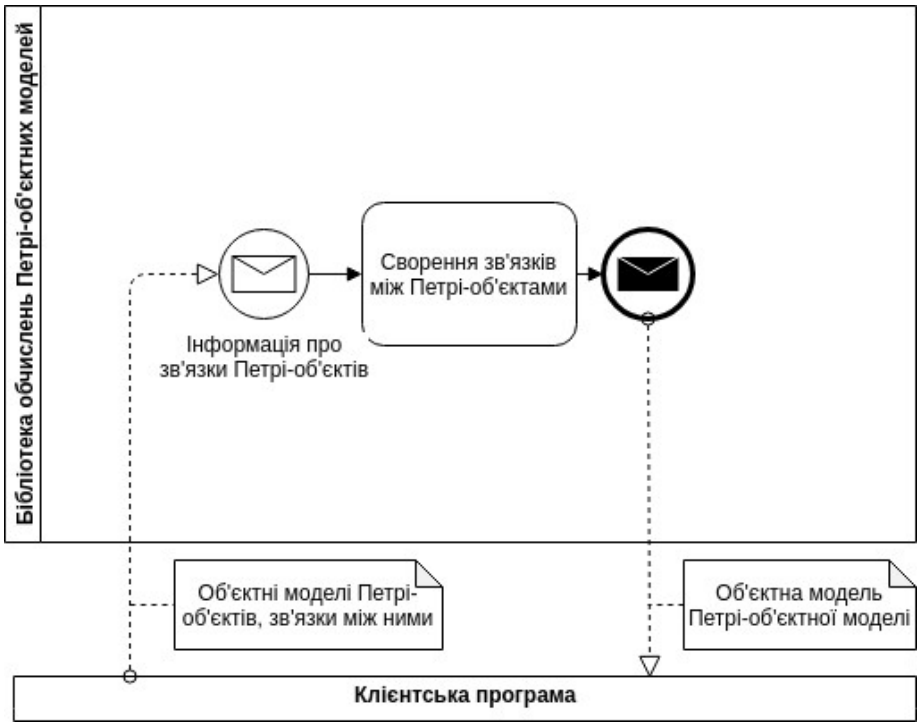


Рисунок 2.2 – Схема бізнес-процесу створення Петрі-об’єктної моделі

Схема бізнес-процесу послідовної імітації Петрі-об’єктної моделі зображена на рисунку 2.3.

Опис бізнес-процесу послідовної імітації Петрі-об’єктної моделі:

– програміст передає бібліотеці об’єктну модель Петрі-об’єктної моделі та час імітації;

– бібліотека виконує валідацію Петрі-об’єктну модель та повідомляє про помилку, якщо вона виникла;

– бібліотека перетворює об’єктну модель Петрі-об’єктної моделі в імітаційну модель;

– бібліотека виконує кроки імітації Петрі-об’єктної моделі, на кожному з яких повідомляє про всі події, що виникли;

– бібліотека повертає статистику імітації.

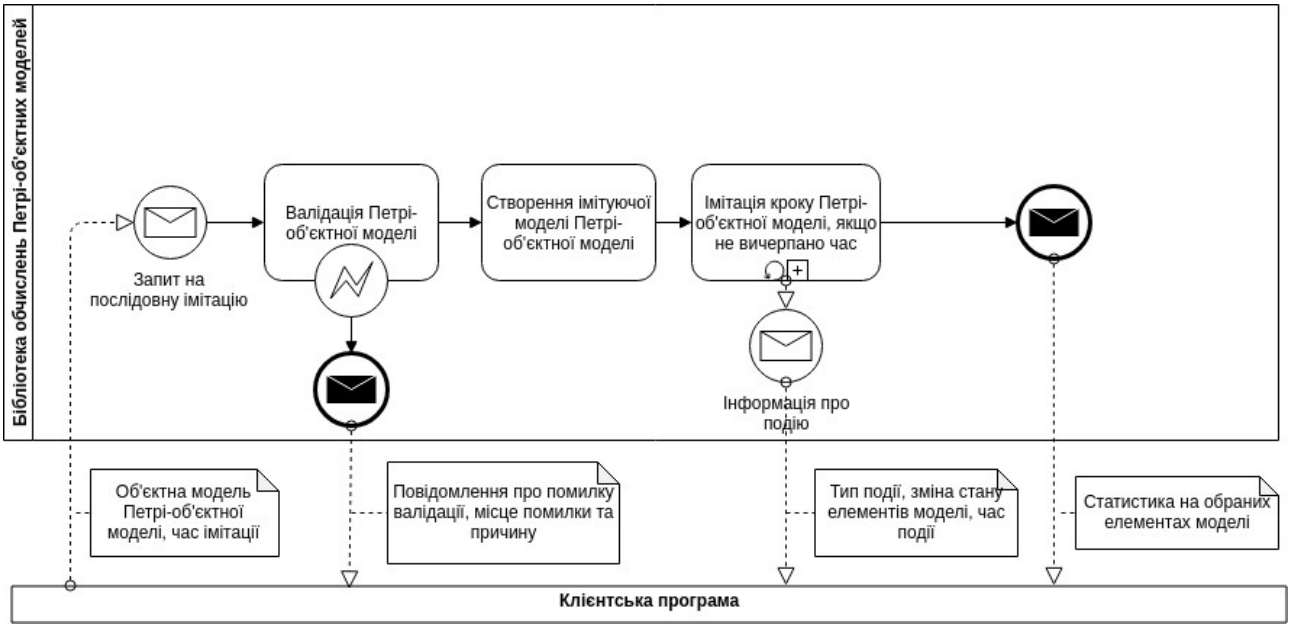


Рисунок 2.3 – Схема бізнес-процесу послідовної імітації Петрі-об’єктної моделі

Схема бізнес-процесу паралельної імітації Петрі-об’єктної моделі зображена на рисунку 2.4.

Опис бізнес-процесу паралельної імітації Петрі-об’єктної моделі:

– програміст передає бібліотеці об’єктну модель Петрі-об’єктної моделі та час імітації;

– бібліотека виконує валідацію Петрі-об’єктну модель та повідомляє про помилку, якщо вона виникла;

– бібліотека перетворює об’єктну модель Петрі-об’єктної моделі в імітаційну модель, розв’язуючи при цьому цикли Петрі-об’єктів;

– бібліотека паралельно виконує кроки імітації Петрі-об’єктів, на кожному з яких кожен Петрі-об’єкт повідомляє про всі події, що виникли через свій окремий інтерфейс;

– бібліотека повертає статистику імітації.

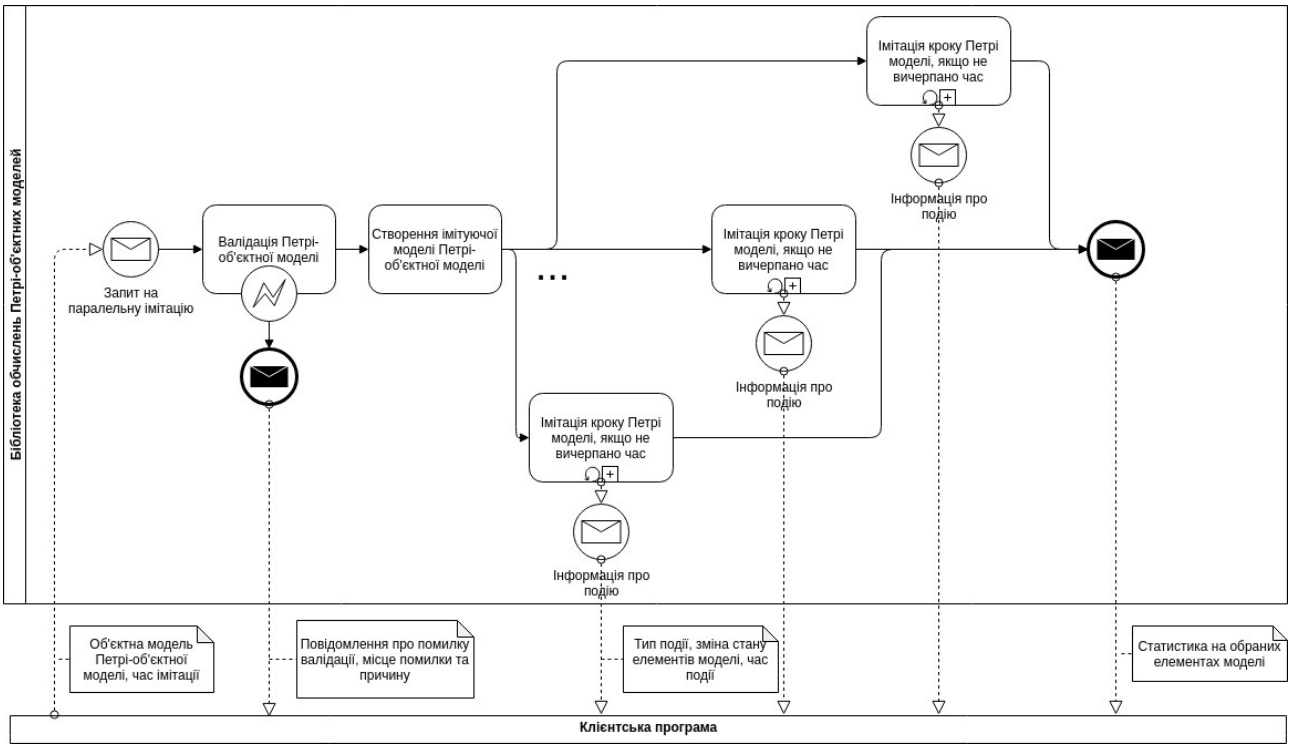


Рисунок 2.4 – Схема бізнес-процесу паралельної імітації Петрі-об’єктної моделі

2.2 Архітектура програмного забезпечення

Для більш гнучкої архітектури бібліотеки її було розділено на п’ять модулів, як зображено в документі КПІ.ІП-5117.045490.06.99.СС “Схема структурна компонентів програмного забезпечення”.

Модуль Model — виділена об’єктна модель, з якою працює Програміст. Це спосіб задання початкових даних алгоритму імітації. Стан цієї моделі не змінюється під час імітації.

Модуль Model computer — модуль для запуску імітації. Цей модуль виконує валідацію моделі та перетворює її в імітаційну модель.

Модуль Event protocol — модуль для роботи з протоколом подій. Тут знаходяться абстракції подій та інтерфейси їх передачі Програмісту.

Модуль Statistics — модуль для роботи зі статистикою. В ньому знаходяться класи, що збирають та організовують статистику для Програміста.

Модуль Computing model — внутрішній модуль бібліотеки, з яким програміст не має працювати напряму. Це ядро бібліотеки, яке виконує паралельне або послідовне обчислювання імітаційної моделі.

Діаграми класів компонентів зображено в документі КПІ.ІП-5117.045490.06.99.СС “Схема структурна класів програмного забезпечення”.

Для задоволення вимог про паралелізм, реалізований на пулі потоків, які перемикаються на імітацію іншого вільного Петрі-об’єкта, якщо імітація їх Петрі-об’єкта, вирішено використати фреймворк Fork/Join мови Java, який дозволяє розбити велику задачу на декілька менших та виконати їх паралельно, автоматично перемикаючи потоки між задачами, коли виникло блокування.

2.3 Конструювання програмного забезпечення

Детальний опис публічних методів класів компонента ModelComputer наведено в таблицях 2.1, 2.2, 2.3.

Таблиця 2.1 — Публічні методи класу PetriObjectModelComputer

Назва методу	Аргументи	Значення, що повертається	Призначення методу
computeSequential	petriObjectModel, timeModeling	-	Перетворення об’єктної моделі Петрі-об’єктної моделі на обчислювальну модель та запуск імітації послідовно на час timeModeling

Продовження таблиці 2.1

computeParallel	petriObjectModel, timeModeling	-	Розплутування циклів Петрі-об’єктної моделі, перетворення її на обчислювальну модель та запуск імітації паралельно на час timeModeling

Таблиця 2.2 — Публічні методи класу StronglyConnectedComponents Resolver

Назва методу	Аргументи	Значення, що повертається	Призначення методу
resolveStrongly Connected Components	petriObject Model	petriObject Model	Перетворювання об’єктної моделі Петрі-об’єктної моделі на таку, у якій компоненти сильної зв’язності графа, утвореним Петрі-об’єктами замінені на Петрі-об’єкти

Таблиця 2.3 — Публічні методи класу PetriObjectMerger

Назва методу	Аргументи	Значення, що повертається	Призначення методу
mergePetri Objects	petriObjects	PetriObject	Створення зі списку Петрі-об’єктів одного Петрі-об’єкта

Детальний опис публічних методів класів компонента Model наведено в таблицях 2.4, 2.5, 2.6, 2.7.

Таблиця 2.4 – Публічні методи класу PetriObjectModel

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
iterator	-	Iterator<PetriObject>	Отримання ітератору, за допомогою якого можна перебирати Петрі-об’єкти моделі

Таблиця 2.5 – Публічні методи класу PetriObject

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
iterator	-	Iterator<Transition>	Отпимання ітератору, за допомогою якого можна перебирати переходи Петрі-об’єкта

Таблиця 2.6 – Публічні методи класу Transition

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
addArcFrom	place, multiplicity	-	Додавання до переходу вхідної дуги від позиції place з кратністю multiplicity
addArcFrom	place	-	Додавання до переходу вхідної дуги від позиції place з кратністю 1
addInformational ArcFrom	place, multiplicity	-	Додавання до переходу інформаційної вхідної дуги від позиції place з кратністю multiplicity

Продовження таблиці 2.6

addInformational ArcFrom	place	-	Додавання до переходу інформаційної вхідної дуги від позиції place з кратністю 1
addArcTo	place, multiplicity	-	Додавання до переходу вихідної дуги в позицію place з кратністю multiplicity
addArcTo	place	-	Додавання до переходу вихідної дуги в позицію place з кратністю 1
setPetriObject	petriObject	-	Задання переходу Петрі-об’єкта, до якого належить цей перехід. Програміст не має викликати цей метод, адже це робиться автоматично при створенні Петрі-об’єкта
resetPetriObjects	-	-	Скидання заданого для переходу Петрі-об’єкта. Цей метод подрібен при об’єднанні компонента сильної зв’язності в більший Петрі-об’єкт. Програміст не має викликати цей метод
isCollectStatistics	-	isCollect Statistics	Отримання інформації про те, чи потрібно збирати статистику для цього переходу

Продовження таблиці 2.6

validate	-	-	Валідування переходу. Програміст не має викликати цей метод, адже він викликається автоматично при створенні Петрі-об’єктної моделі

Таблиця 2.7 – Публічні методи класу Place

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
setPetriObject	petriObject	-	Задання переходу Петрі-об’єкта, до якого належить ця позиція. Програміст не має викликати цей метод, адже це робиться автоматично при створенні Петрі-об’єкта
resetPetriObject	-	-	Скидання заданого для позиції Петрі-об’єкта. Цей метод потрібен при об’єднанні компонента сильної зв’язності в більший Петрі-об’єкт. Програміст не має викликати цей метод
isCollectStatistics	-	isCollectStatistics	Отримання інформації про те, чи потрібно збирати статистику для цієї позиції

Детальний опис публічних методів класів компонента Statistics наведено в таблицях 2.8, 2.9, 2.10, 2.11.

Таблиця 2.8 – Публічні методи класу PetriObjectModelStatistics

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
getPetriObject StatisticsBy PetriObjectId	petriObjectId	PetriObjectStatistics	Отримання статистики Петрі-об’єкта за його petriObjectId
doStatistics	timeDelta	-	Підрахунок статистики за час timeDelta. Програміст не має викликати цей метод, адже це робиться автоматично під час імітації

Таблиця 2.9 – Публічні методи класу PetriObjectStatistics

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
getTransition StatisticsBy TransitionId	transitionId	TransitionStatistics	Отримання статистики переходу за його transitionId
getPlace StatisticsBy PlaceId	placeId	PlaceStatistics	Отримання статистики позиції за її placeId
doStatistics	timeDelta	-	Підрахунок статистики за час timeDelta. Програміст не має викликати цей метод, адже це робиться автоматично під час імітації

Таблиця 2.10 – Публічні методи класу TransitionStatistics

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
getBufferSize	-	bufferSize	Отримання кількості подій в буфері на момент завершення імітації
getMaxObserved BufferSize	-	maxObserved BufferSize	Отримання максимальної кількості подій в буфері під час імітації
getMinObserved BufferSize	-	minObserved BufferSize	Отримання мінімальної кількості подій в буфері під час імітації
getMeanBuffer Size	-	meanBuffer Size	Отримання середньої кількості подій в буфері під час імітації
doStatistics	timeDelta	-	Підрахунок статистики за час timeDelta. Програміст не має викликати цей метод, адже це робиться автоматично під час імітації

Таблиця 2.11 – Публічні методи класу PlaceStatistics

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
getMarking	-	marking	Отримання маркування позиції на момент завершення імітації
getMaxObserved Marking	-	maxObserved Marking	Отримання максимального маркування позиції під час імітації
getMinObserved Marking	-	minObserved Marking	Отримання мінімального маркування позиції під час імітації

Продовження таблиці 2.11

getMeanMarking	-	meanMarking	Отримання середнього маркування позиції під час імітації
doStatistics	timeDelta	-	Підрахунок статистики за час timeDelta. Програміст не має викликати цей метод, адже це робиться автоматично під час імітації

Детальний опис методів інтерфейсів модуля EventProtocol, які Програміст має реалізувати у своїй програмі наведено в таблицях 2.12, 2.13.

Таблиця 2.12 – Методи інтерфейсу EventProtocol

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
isEnabled	-	isEnabled	Отримання інформації про те, чи потрібно передавати події в протокол
onTimeMoved Event	timeMoved Event	-	Повідомлення протоколу про просування часу
onTransition ActInCompleted Event	transition ActInCompleted Event	-	Повідомлення протоколу про вхід маркерів в перехід

Продовження таблиці 2.12

onPetriObject ActOutCompleted Event	petriObject ActOutCompleted Event	-	Повідомлення протоколу про вихід маркерів з переходів на всьому Петрі-об’єкті
onPetriObject ActInCompleted Event	petriObject ActInCompleted Event	-	Повідомлення протоколу про вхід маркерів в переходи на всьому Петрі-об’єкті
onPetriObject ModelActOut CompletedEvent	petriObject ModelActOut CompletedEvent	-	Повідомлення протоколу про вихід маркерів з переходів на всій Петрі-об’єктній моделі
onPetriObject ModelActIn CompletedEvent	petriObjectModel ActInCompleted Event	-	Повідомлення протоколу про вхід маркерів в переходи на всій Петрі-об’єктній моделі
onPetriObjects ConflictResolved Event	petriObjects ConflictResolved Event	-	Повідомлення протоколу про розв’язання конфлікту між Петрі-об’єктами

Продовження таблиці 2.12

onTransitions ConflictResolved Event	transitionsConflict ResolvedEvent	-	Повідомлення протоколу про розв’язання конфлікту між переходами

Таблиця 2.13 – Методи інтерфейсу EventProtocolFactory

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
createEventProtocol	-	EventProtocol	Створення нового протоколу подій

Програміст не повинен напряму працювати з класами компонента ComputingModel, але для простішої підтримки бібліотеки основні методи цих класів наведено в таблицях 2.14, 2.15, 2.16, 2.17, 2.18, 2.19, 2.20, 2.21, 2.22, 2.23, 2.24.

Таблиця 2.14 – Основні методи класу ParallelCoomputingPetriObjectModel

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
go	simulationTime	-	Запуск імітації Петрі-об’єктів на Fork/Join пулі потоків та очікування завершення імітації кожного Петрі-об’єкта

Таблиця 2.15 – Основні методи класу ParallelComputingPetriObject

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
run	-	-	Запуск імітації Петрі-об’єкта в потоці
moveTime	-	-	Просування локального часу Петрі-об’єкта

Продовження таблиці 2.15

defineNearest ExternalEvent Time	-	-	Визначення найближчого часу зовнішньої події від усіх Петрі-об’єктів, які надсилають повідомлення у цей Петрі-об’єкт. Очікування, якщо є порожні буфери повідомлень
actExternalOut	currentTime	-	Отримання повідомлень з буферів, якщо її часова позначка дорівнює currentTime
awaitIf ReachedLimit	-	-	Очікування при умові, що розмір найменшого буфера повідомлень, у який цей об’єкт надсилає повідомлення, дорівнює або перевищує заданий ліміт
signalEvent Taken FromNext ObjectBuffer	-	-	Повідомити Петрі-об’єкт, що об’єкт, у який даний об’єкт надсилає повідомлення, забрав повідомлення із свого буфера
getMin External BufferSize	-	minExternal BufferSize	Отримати розмір найменшого буфера повідомлень, у який цей об’єкт надсилає повідомлення
setTimeState	timeState	-	Задання стану часу з поточним часом та часом імітації
addExternal ArcOut	externalArc Out	-	Додавання до Петрі-об’єкта зовнішньої дуги до іншого об’єкта, через яку даний об’єкт надсилає повідомлення

Таблиця 2.16 – Основні методи класу ComputingExternalArcOut

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
getNearest EventTime	-	nearestEventTime	Отримання найближчого часу повідомлення від Петрі-об’єкта, або очікування поки повідомлення не надійде
runArc	currentTime	-	Надіслати повідомлення до Петрі-об’єкта з часовою позначкою currentTime
actOut	currentTime	-	Забрати повідомлення з буфера, часова позначка якого дорівнює currentTime, повідомити про це попередній Петрі-об’єкт та збільшити маркування позиції, в яку направлена дуга
getBuffer Size	-	bufferSize	Отримання кількості повідомлень в буфері

Таблиця 2.17 – Основні методи класу SequentialComputingPetriObject Model

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
go	simulation Time	-	Запуск послідовної імітації Петрі-об’єктної моделі на час імітації simulationTime
moveTime	-	-	Просування часу моделі до часу найближчої події на Петрі-об’єктах

Продовження таблиці 2.17

actIn	currentTime	-	Запуск входу в переходи для Петрі-об’єктів, які мають активні переходи, у порядку розв’язання конфліктів
actOut	currentTime	-	Запуск виходу подій з переходів на Петрі-об’єктах, час найближчої події на яких дорівнює currentTime
resolveConflict	activePetri Objects	petriObject	Розв’язання конфлікту декількох Петрі-об’єктів, кожен з яких може виконати вхід в переходи
select Prioritised PetriObjects	petriObjects	prioritised PetriObjects	Вибірка з Петрі-об’єктів Петрі-об’єктів з найбільшим пріоритетом
selectRandom PetriObject	petriObjects	petriObject	Вибір з Петрі-об’єктів випадкового
getActivePetri Objects	-	activePetri Objects	Отримання списку Петрі-об’єктів, для яких може бути виконаним вхід в переходи

Таблиця 2.18 – Основні методи класу SequentialComputingPetriObject

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
actIn	currentTime	-	Запуск входу в активні переходи у порядку розв’язання конфліктів
actOut	currentTime	-	Запуск виходу подій з переходів, час найближчої події на яких дорівнює currentTime

Продовження таблиці 2.18

getActive Transitions	-	activeTransitions	Отримання списку переходів, для яких може бути виконаним вхід
resolveConflict	active Transitions	transition	Розв’язання конфлікту декількох переходів, для яких може бути виконаним вхід
selectPrioritised Transitions	transitions	prioritisedTransitions	Вибірка з переходів переходів з найбільшим пріоритетом
selectRandom Transition	transitions	transition	Вибір з переходів випадкового з урахуванням їх вказаної імовірності
defineNearest EventTime	-	-	Визначення часу найближчої події в Петрі-об’єкті
isActive	-	isActive	Визначення, чи є в Петрі-об’єкти активні переходи

Таблиця 2.19 – Основні методи класу ComputingTransition

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
addArcIn	arcIn	-	Додавання до переходу вхідної дуги
addAllArcsIn	arcsIn	-	Додавання до переходу декількох вхідних дуг
addArcOut	arcOut	-	Додавання до переходу вихідної дуги

Продовження таблиці 2.19

addAllArcsOut	arcsOut	-	Додавання до переходу декількох вихідних дуг
isActive	-	isActive	Визначення, чи може бути виконаний вхід у перехід
actIn	currentTime	-	Виконання входу у перехід
actOut	currentTime	-	Виконання виходу з переходу
getNearest EventTime	-	nearestEventTime	Отримання часу найближчої події в переході

Таблиця 2.20 – Основні методи класу EventTimeBuffer

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
getNearest EventTime	-	nearestEventTime	Отримання часу найближчої події в буфері
addEvent	eventTime	-	Додавання події до буфера та перевизначення часу найближчої події в буфері
removeNearest Event	-	-	Прибирання найближчої події з буфера та перевизначення часу найближчої події в буфері
defineNearest EventTime	-	-	Визначення часу найближчої події в буфері

Таблиця 2.21 – Основні методи класу ComputingPlace

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
increaseMarking	delta	-	Збільшення маркування позиції на delta
decreaseMarking	delta	-	Зменшення маркування позиції на delta

Таблиця 2.22 – Основні методи класу ComputingArcIn

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
isRunnable	-	isRunnable	Перевірка, чи більша або дорівнює маркування позиції, з якої виходить дуга, за кратність дуги
runArc	currentTime	-	Зменшення маркування позиції, з якої виходить дуга, на кратність дуги

Таблиця 2.23 – Основні методи класу ComputingInformationalArcIn

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
runArc	currentTime	-	Валідація, чи більша або дорівнює маркування позиції, з якої виходить дуга, за кратність дуги

Таблиця 2.24 – Основні методи класу ComputingArcOut

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
runArc	currentTime	-	Збільшення маркування позиції, в яку входить дуга, на кратність дуги

2.4 Опис вхідних даних

На вхід бібліотеки паралельної імітації Петрі-об’єктних моделей необхідно подавати об’єктну модель Петрі-об’єктної моделі. Параметри створення Петрі-об’єктної моделі з програми, у яку інтегрували бібліотеку, наведено в таблиці 2.25.

Таблиця 2.25 – Параметри Петрі-об’єктної моделі

Назва параметра	Тип параметра	Обов’язковий	Значення за замовчуванням	Опис параметра
petriObjects	List<PetriObject>	Так	-	Список Петрі-об’єктів

Параметри створення Петрі-об’єкта наведено в таблиці 2.26.

Таблиця 2.26 – Параметри Петрі-об'єкта

Назва параметра	Тип параметра	Обов’язковий	Значення за замовчуванням	Опис параметра
petriObjectId	long	Так	-	Унікальний ідентифікатор Петрі-об’єкта
petriObject Name	String	Так	-	Назва Петрі-об’єкта
transitions	List<Transition>	Так	-	список об’єктних моделей переходів цього Петрі-об’єкта
priority	int	Ні	1	Значення пріоритету цього Петрі-об’єкта

Продовження таблиці 2.26

externalBuffer SizeLimit	int	Hi	2147483647	значення обмеження розміру мінімального буфера вихідних подій

Параметри створення об’єктної моделі переходу наведено в таблиці 2.19.

Таблиця 2.27 – Параметри переходу

Назва параметра	Тип параметра	Обов’язковий	Значення за замовчуванням	Опис параметра
transitionId	long	Так	-	Унікальний ідентифікатор переходу
transition Name	String	Так	-	Назва переходу
priority	Int	Hi	1	Значення пріоритету запуска цього переходу при виникненні конфлікту
probability	double	Hi	1	значення ймовірності запуска цього переходу при виникненні конфлікту

Продовження таблиці 2.27

delay Generator	DelayGenerator	Hi	Константна затримка 0	генератор затримки в переході
collect Statistics	boolean	Hi	true	Флаг, який показує чи потрібно збирати статистику для цього переходу

Після того, як перехід створено, його потрібно об’єднати з позиціями за допомогою методів на об’єктній моделі переходу. Проведення вхідної дуги з програми: «transition.addArcFrom(place, multiplicity);», де place — об’єктна модель позиції, multiplicity — опціональне цілочисельне значення кратності дуги (за замовчуванням 1). Проведення вхідної інформаційної дуги з програми: «transition.addInformationalArcFrom(place, multiplicity);», де place — об’єктна модель позиції, multiplicity — опціональне цілочисельне значення кратності дуги (за замовчуванням 1). Проведення вихідної дуги з програми: «transition.addArcTo(place, multiplicity);», де place — об’єктна модель позиції, multiplicity — опціональне цілочисельне значення кратності дуги (за замовчуванням 1).

Параметри створення об’єктної моделі позиції наведено в таблиці 2.28.

Таблиця 2.28 – Параметри позиції

Назва параметра	Тип параметра	Обов’язковий	Значення за замовчуванням	Опис параметра
placeId	long	Так	-	Унікальний ідентифікатор позиції
placeName	String	Так	-	Назва позиції
marking	int	Hi	0	Значення початкового маркування позиції

Продовження таблиці 2.28

collect Statistics	boolean	Hi	true	Флаг, який показує чи потрібно збирати статистику для цієї позиції

2.5 Опис вихідних даних

Бібліотека паралельних обчислень Петрі-об’єктних моделей повертає два типи даних: протокол подій та статистику.

Для того, щоб отримувати протокол подій необхідно передати бібліотеці свою реалізацію інтерфейсу EventProtocol для послідовної імітації або свою реалізацію інтерфейсу EventProtocolFactory, яка має повертати реалізацію EventProtocol, для паралельної імітації. В EventProtocol передаються об’єктні моделі всіх подій, які мають час, у який відбулася ця подія, та необхідна інформація про стан моделі після події.

Об’єктну модель статистики Петрі-об’єктної моделі повертає клас PetriObjectModelComputer. Для того, щоб отримати статистику конкретного переходу чи позиції, необхідно зі статистики Петрі-об’єктної моделі дістати статистику Петрі-об’єкта за чисельним ідентифікатором цього Петрі-об’єкта, а потім за чисельним ідентифікатором переходу чи позиції отримати зі статистики Петрі-об’єкта статистику переходу чи позиції відповідно.

Статистика переходу містить в собі інформацію про кількість подій в переході на момент завершення імітації, максимальну, мінімальну та середню кількість подій в переході за весь час імітації.

Статистика позиції містить в собі інформацію про маркування позиції на момент завершення імітації, максимальне, мінімальне та середнє маркування позиції за весь час імітації.

2.6 Аналіз безпеки даних

Розроблений програмний продукт ні в якому вигляді не зберігає дані та не передає їх через мережу. Тому використання алгоритмів шифрування, хеш-функцій або захищених протоколів передачі даних не потребується.

Крім того, імітація Петрі-об’єктної моделі не припускає наявність персональних чи вразливих даних.

2.7 Висновки по розділу

У цьому розділі для бібліотеки паралельної імітації Перті-об’єктних моделей було виділено бізнес-процеси з оглядом на сценарії використання, та представлено їх у вигляді BPMN діаграм.

Також була розроблена архітектура бібліотеки, представлена у вигляді діаграм компонентів та класів, і описані основні методи класів.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Оцінити якість програмного забезпечення для обчислення Петрі-об’єктних моделей можна декількома способами:

– порівняти результати роботи програми з аналітичними розрахунками;

– порівняти результати роботи програми з результатами іншої налагодженої програми;

– порівняти результати роботи послідовного і паралельного алгоритмів бібліотеки.

Результати роботи можуть відрізнятися один від одного, адже в алгоритмі імітації є елемент випадковості, тому тестування необхідно проводити декілька разів та аналізувати середні результати прогонів. В такому разі результати аналітичних розрахунків та роботи алгоритму або результати роботи двох алгоритмів мають відрізнятися на дуже малий відсоток.

3.2 Опис процесів тестування

В першу чергу необхідно порівняти результати роботи алгоритму з аналітичними розрахунками. Згідно з розрахунками в результаті імітації Петрі мережі зображеної на малюнку 3.1 мають бути отримані такі середні значення маркування позицій: : Queue1 — 1,786, Queue2 — 0,003, Queue3 — 0,004, Queue4 — 0,00001, Limit 1 — 0,268, Limit 2 — 0,946, Limit 3 — 0,938, Limit 4 — 1.964.

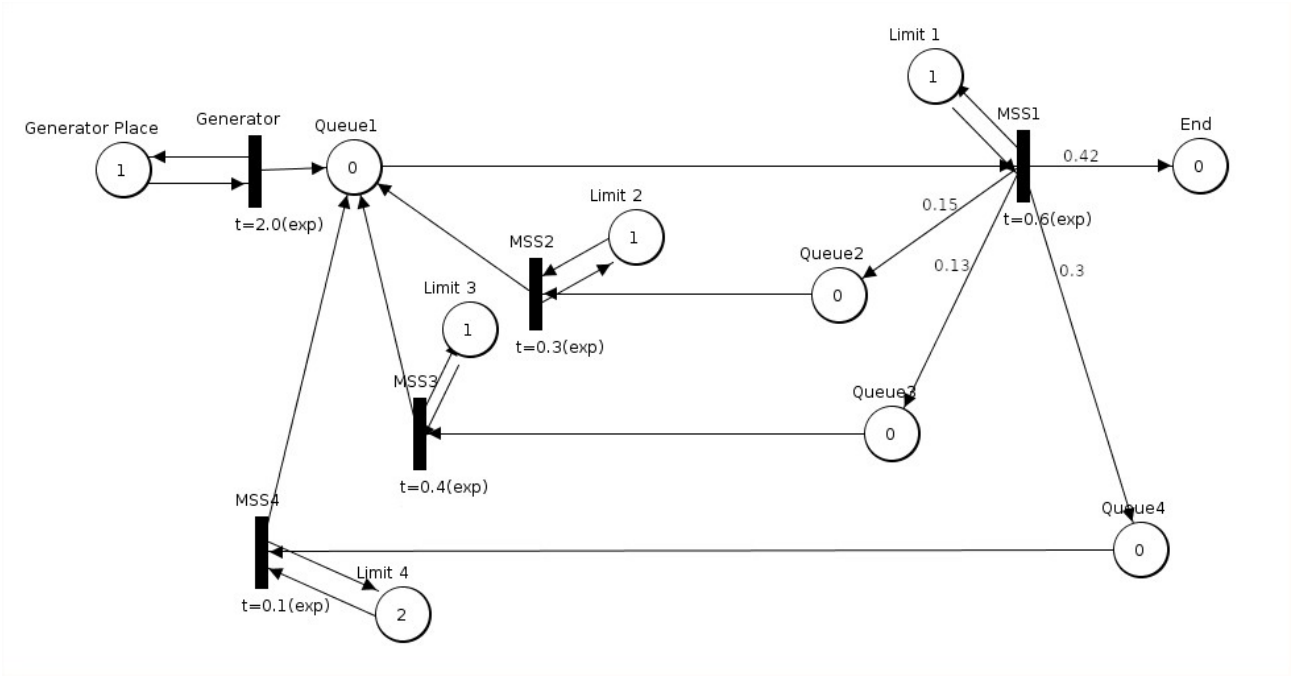


Рисунок 3.1 — Петрі мережа для аналітичних розрахунків

Бібліотека паралельних обчислень Петрі-об’єктних моделей запущена на час імітації 1000000 одиниць на цій мережі повернула такі результати:

Queue1 - 1.8135105843382364

Queue2 - 0.003047752229117642

Queue3 - 0.004201189608231018

Queue4 - 1.0507239076957115E-5

Limit1 - 0.2838262990831258

Limit2 - 0.9463307505731481

Limit3 - 0.9376207085139558

Limit4 - 1.964196857390901

Отже, точність розробленого алгоритму – 97,6%. Таким чином далі можна порівнювати результати роботи послідовного та паралельного алгоритмів на однакових Петрі-об’єктних моделях, та тестувати з оглядом на їх результати.

3.3 Опис контрольного прикладу

Для порівняння швидкодії послідовного та паралельного алгоритмів імітації було створено набір Петрі-об’єктних моделей за схемою зображеною на рисунку 3.2.

Рисунок 3.2 – Схема Петрі-об’єктної мережі для порівняння часу роботи послідовного та паралельного алгоритмів

Першим до Петрі-об’єктної моделі додається Петрі-об’єкт генератора, а потім від п’яти до п’ятдесяти з кроком п’ять Петрі-об’єктів, які складаються з десяти СМО. Кожен перехід, що відображає СМО виконує вихід маркерів у чергу наступної СМО. Таким чином, загальна кількість переходів змінюється від п’ятдесяти до п’ятисота з кроком п’ятдесят. Результати експериментів наведені у вигляді графіка на рисунку 3.3.

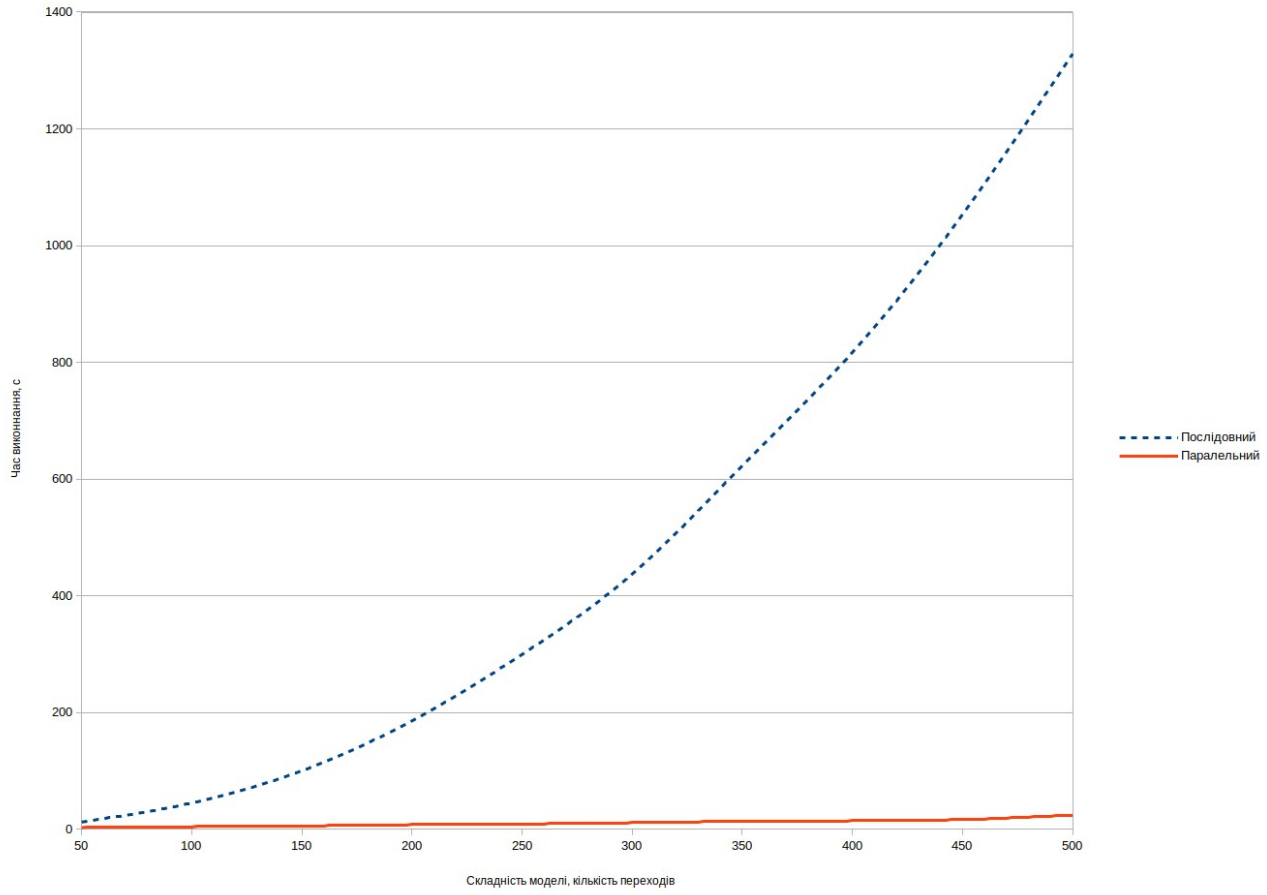


Рисунок 3.3 – Результати експерименту порівняння швидкодії послідовного та паралельного алгоритмів

З графіка видно, що для побудованої Петрі-об’єктної моделі час роботи послідовного алгоритму кубічно залежить від кількості переходів, а час роботи паралельного алгоритму – лінійно. Для п’ятисот переходів паралельний алгоритм працює швидше послідовного у п’ятдесят п’ять разів.

Експеримент проводився на обладнанні з процесором Intel Core i7 сьомого покоління та з об’ємом оперативної пам’яті 32 Гб.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Бібліотека паралельних обчислень Петрі-об’єктних моделей розроблена для інтеграції в інші програми, написані мовою Java.

Для того, щоб інтегрувати бібліотеку, необхідно включати необхідні класи у файли програми, де ці класи використовуються, за допомогою ключового слова import, за яким має слідувати повне ім’я класу та крапка з комою.

При запуску програми, у яку інтегрована бібліотека паралельних обчислень Петрі-об’єктних моделей, необхідно додати jar-файл бібліотеки до шляху до класів за допомогою параметра -cp команди “java”, яка запускає програми написані мовою Java.

4.2 Робота з програмним забезпеченням

Детальний опис роботи з бібліотекою паралельних обчислень Петрі-об’єктних моделей наведено у документі КПІ.ІП-5117.045490.05.34 “Керівництво користувача”.

4.3 Висновки по розділу

В цьому розділі описано процес інтеграцію бібліотеки паралельних обчислень Петрі-об’єктних моделей до іншої програми написаною мовою Java.

ВИСНОВКИ

У цьому дипломному проекті проаналізовані підходи до паралельної імітації моделей, а зокрема паралельний алгоритм імітації Петрі-об’єктних моделей, та сформульовані вимоги до бібліотеки паралельних обчислень Петрі-об’єктних моделей.

Спроектована і розроблена програмна бібліотека мовою Java для інтеграції в інші програмні продукти для імітації моделей. Створена бібліотека має ряд переваг:

– користувачеві не потрібно змінювати Петрі-об’єктну модель для послідовної та паралельної імітації, а необхідне приведення до форми, що вимагає алгоритм, робиться автоматично;

– при великій кількості компонентів сильної зв’язності в Петрі-об’єктній моделі імітація відбувається у десятки разів швидше;

– бібліотека підтримує різні способи задання затримках в переходах, а також інформаційні вхідні дуги.

Проаналізована та протестована отримана бібліотека і налагоджені знайдені помилки.

Розроблена проектна документація, структурні схеми варіантів використання, компонентів та класів програмного забезпечення, а також керівництво користувача.

ПЕРЕЛІК ПОСИЛАНЬ

1) Stetsenko Inna V., Dorosh Vitaliy I., Dyfuchyn Anton Petri-object sumulation: software package and compexite // Intelligent Data Acquisition and Andvanced Computing Systems: Technology and Applications (IDAACS), 2015 IEEE 8th International Conference. - IEEE, 2015. - Vol.1. - С. 381-385.

2) Jason L. Parallel Discrete-Event Simulation / Liu Jason., 2010.

3) Fujimoto Richard M. Parallel and distributed simulation // Proceedings of the 2015 Winter Simulation Conference. - IEEE, 2015. - С. 47-52.

4) Стеценко І.В. Паралельний алгоритм імітації Петрі-об'єктної моделі / Інна Вячеславівна Стеценко. // Кібернетика і системний аналіз. - 2017. - №53(4). - С. 130-140.

5) Стеценко І.В. Алгоритм імітації Петрі-об'єктної моделі // Математичні машини і системи. – Київ, 2012. – №1. – С.154-165.

6) Foster I. Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. – Addison-Wesley Longman Publishing Co., 1995 – 370 с.

7) Garg Vijay K. Concurrent and distributed computing in Java. – John Willey & sons, Inc., 2004 – 305с.

8) Fujimoto R. M. Parallel and distributed simulation system // Proceedings of the 2001 Winter Simulation Conference – С.147-156.

9) W.M.P. van der Aalst. Process-Aware information Systems: Lessons ti be Learned from Process Mining // Transactions on Petri Nets and Other Models of Concurrency II. Special Issue on Concurrency in Process-Aware Information Systems. – Springer, 2009. – С.1-26.

10) W.M.P. van der Aalst. Business Process Management as the "Killer App" for Petri Nets // Software and System Modeling. – 2015. С.1-20.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2019 р.

БІБЛІОТЕКА ПАРАЛЕЛЬНИХ ОБЧИСЛЕНІ ПЕТРІ-ОБ’ЄКТНИХ МОДЕЛЕЙ

Опис програми

КПІ.ІП-5117.045490.02.13

“ПОГОДЖЕНО”

Керівник проекту:

_____ І.В. Стеценко

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____ А.В. Педоренко

Київ – 2019 року

Тексти програмного коду

Бібліотека паралельних обчислень Петрі-об'єктних моделей

(Найменування програми (документа))

CD-RW

(Вид носія даних)

13 арк, 159 Кб

(Обсяг програми (документа) , арк.,)

Київ - 2019

Модуль Model.

Клас PetriObjectModel.

```
package edu.pedorenko.petri_object_model.model;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
public class PetriObjectModel implements Iterable<PetriObject> {
    private List<PetriObject> petriObjects;
    public PetriObjectModel(List<PetriObject> petriObjects) throws
PetriObjectModelException {
        this.petriObjects = petriObjects;
        validate();
    }
    private void validate() throws PetriObjectModelException {
        Set<Long> petriObjectIdsSet = new HashSet<>();
        Set<Long> transitionIdsSet = new HashSet<>();
        Set<Long> placeIdsSet = new HashSet<>();
        for (PetriObject petriObject : petriObjects) {
            for (Long petriObjectId : petriObject.getPetriObjectIds()) {
                if (petriObjectIdsSet.contains(petriObjectId)) {
                    throw new PetriObjectModelException("Duplicate
petriObjectId=" + petriObjectId + " found in petri object model");
                }
                petriObjectIdsSet.add(petriObjectId);
            }
            petriObject.validate(transitionIdsSet, placeIdsSet);
        }
    }
    public Iterator<PetriObject> iterator() {
        return petriObjects.iterator();
    }
}
```

Клас PetriObject.

```
package edu.pedorenko.petri_object_model.model;
import edu.pedorenko.petri_object_model.model.arc.ArcIn;
import edu.pedorenko.petri_object_model.model.arc.ArcOut;
import edu.pedorenko.petri_object_model.model.place.Place;
import edu.pedorenko.petri_object_model.model.transition.Transition;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;
public class PetriObject implements Iterable<Transition> {
    private List<Long> petriObjectIds; //several petri object can be
connected to one with an algorithm
    private String petriObjectName;
    private Map<Long, Transition> transitionsMap = new HashMap<>();
    private Map<Long, Place> placesMap = new HashMap<>();
    private int priority;
    private int externalBufferSizeLimit;
```

```
    public PetriObject(
            List<Long> petriObjectIds,
            String petriObjectName,
            List<Transition> transitions,
            int priority,
            int externalBufferSizeLimit) throws
PetriObjectModelException {
        this.petriObjectIds = petriObjectIds;
        this.petriObjectName = petriObjectName;
        this.priority = priority;
        this.externalBufferSizeLimit = externalBufferSizeLimit;
        fillMaps(transitions);
    }
    public PetriObject(
            long petriObjectId,
            String petriObjectName,
            List<Transition> transitions,
            int priority,
            int externalBufferSizeLimit) throws
PetriObjectModelException {
        this(new ArrayList<Long>(){{add(petriObjectId);}},
petriObjectName, transitions, priority, externalBufferSizeLimit);
    }
    public PetriObject(long petriObjectId, String petriObjectName,
List<Transition> transitions, int priority) throws
PetriObjectModelException {
        this(petriObjectId, petriObjectName, transitions, priority,
Integer.MAX_VALUE);
    }
    public PetriObject(long petriObjectId, String petriObjectName,
List<Transition> transitions) throws PetriObjectModelException {
        this(petriObjectId, petriObjectName, transitions, 1);
    }
    public PetriObject(long petriObjectId, String petriObjectName, int
externalBufferSizeLimit, List<Transition> transitions) throws
PetriObjectModelException {
        this(petriObjectId, petriObjectName, transitions, 1,
externalBufferSizeLimit);
    }
    private void fillMaps(List<Transition> transitions) throws
PetriObjectModelException {
        for (Transition transition : transitions) {
            for (ArcIn arcIn : transition.getArcsIn()) {
                Place place = arcIn.getPlace();
                addPlace(place);
            }
            for (ArcOut arcOut : transition.getArcsOut()) {
                Place place = arcOut.getPlace();
                addPlace(place);
            }
            addTransition(transition);
        }
    }
    private void addPlace(Place place) throws PetriObjectModelException
{
        long placeId = place.getPlaceId();
        if (placesMap.containsKey(placeId)) {
```

```
            Place placeInMap = placesMap.get(placeId);
            if (placeInMap != place) { //really compare by links
                throw new PetriObjectModelException("Different places
with same id in Petri object.\n" +
                        "Place 1: \n" + placeInMap + "\n" +
                        "Place 2: \n" + place);
            }
        } else {
            placesMap.put(placeId, place);
            place.setPetriObject(this);
        }
    }
    private void addTransition(Transition transition) throws
PetriObjectModelException {
        long transitionId = transition.getTransitionId();
        if (transitionsMap.containsKey(transitionId)) {
            Transition transitionInMap =
transitionsMap.get(transitionId);
            if (transitionInMap != transition) { //really compare by
links
                throw new PetriObjectModelException("Different
transitions with same id in Petri object.\n" +
                        "Transition 1: \n" + transitionInMap + "\n" +
                        "Transition 2: \n" + transition);
            }
        } else {
            transitionsMap.put(transitionId, transition);
            transition.setPetriObject(this);
        }
    }
    public Transition getTransitionById(long transitionId) {
        return transitionsMap.get(transitionId);
    }
    public Place getPlaceById(long placeId) {
        return placesMap.get(placeId);
    }
    public List<Long> getPetriObjectIds() {
        return petriObjectIds;
    }
    public long getPetriObjectId() {
        return petriObjectIds.get(0);
    }
    public String getPetriObjectName() {
        return petriObjectName;
    }
    public int getPriority() {
        return priority;
    }
    public int getExternalBufferSizeLimit() {
        return externalBufferSizeLimit;
    }
    public Collection<Place> getThisObjectPlaces() {
        return placesMap.values();
    }
    public Iterator<Transition> iterator() {
        return transitionsMap.values().iterator();
    }
```

```
    void validate(Set<Long> transitionIdsSet, Set<Long> placeIdsSet)
throws PetriObjectModelException {
        for (Transition transition : transitionsMap.values()) {
            long transitionId = transition.getTransitionId();
            if (transitionIdsSet.contains(transitionId)) {
                throw new PetriObjectModelException("Duplicate
transitionId=" + transitionId + " found." +
                        " Transition ids should be unique event in
different Petri objects");
            }
            transitionIdsSet.add(transitionId);
            transition.validate();
        }
        for (Place place : placesMap.values()) {
            long placeId = place.getPlaceId();
            if (placeIdsSet.contains(placeId)) {
                throw new PetriObjectModelException("Duplicate placeId="
+ placeId + " found." +
                        " Place ids should be unique event in different
Petri objects");
            }
            placeIdsSet.add(placeId);
        }
    }
}
```

Клас PetriObjectModelException.

```
package edu.pedorenko.petri_object_model.model;
public class PetriObjectModelException extends Exception {
    public PetriObjectModelException(String message) {
        super(message);
    }
    public PetriObjectModelException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

Клас Transition.

```
package edu.pedorenko.petri_object_model.model.transition;
import edu.pedorenko.petri_object_model.model.PetriObject;
import edu.pedorenko.petri_object_model.model.PetriObjectModelException;
import
edu.pedorenko.petri_object_model.model.transition.delay_generator.Consta
ntDelayGenerator;
import
edu.pedorenko.petri_object_model.model.transition.delay_generator.DelayG
enerator;
import edu.pedorenko.petri_object_model.model.arc.ArcIn;
import edu.pedorenko.petri_object_model.model.arc.ArcOut;
import edu.pedorenko.petri_object_model.model.arc.InformationalArcIn;
import edu.pedorenko.petri_object_model.model.place.Place;
import java.util.ArrayList;
import java.util.List;
public class Transition {
    private long transitionId;
    private String transitionName;
    private int priority;
```

```
    private double probability;
    private DelayGenerator delayGenerator;
    private List<ArcIn> arcsIn = new ArrayList<>();
    private List<ArcOut> arcsOut = new ArrayList<>();
    private PetriObject petriObject;
    private boolean collectStatistics;
    public Transition(
            long transitionId,
            String transitionName,
            int priority,
            double probability,
            DelayGenerator delayGenerator,
            boolean collectStatistics) {
        this.transitionId = transitionId;
        this.transitionName = transitionName;
        this.priority = priority;
        this.probability = probability;
        this.delayGenerator = delayGenerator;
        this.collectStatistics = collectStatistics;
    }
    public Transition(
            long transitionId,
            String transitionName,
            int priority,
            double probability,
            DelayGenerator delayGenerator) {
        this(
                transitionId,
                transitionName,
                priority,
                probability,
                delayGenerator,
                true);
    }
    public Transition(
            long transitionId,
            String transitionName,
            int priority,
            double probability,
            boolean collectStatistics) {
        this(
                transitionId,
                transitionName,
                priority,
                probability,
                new ConstantDelayGenerator(0),
                collectStatistics);
    }
    public Transition(
            long transitionId,
            String transitionName,
            int priority,
            DelayGenerator delayGenerator,
            boolean collectStatistics) {
        this(
                transitionId, transitionName,
                priority,
```

```
                1d,
                delayGenerator,
                collectStatistics);
    }
    public Transition(
            long transitionId,
            String transitionName,
            double probability,
            DelayGenerator delayGenerator,
            boolean collectStatistics) {
        this(
                transitionId,
                transitionName,
                0,
                probability,
                delayGenerator,
                collectStatistics);
    }
    public Transition(
            long transitionId,
            String transitionName,
            int priority,
            double probability) {
        this(
                transitionId,
                transitionName,
                priority,
                probability,
                new ConstantDelayGenerator(0));
    }
    public Transition(
            long transitionId,
            String transitionName,
            int priority,
            DelayGenerator delayGenerator) {
        this(
                transitionId, transitionName,
                priority,
                1d,
                delayGenerator);
    }
    public Transition(
            long transitionId,
            String transitionName,
            double probability,
            DelayGenerator delayGenerator) {
        this(
                transitionId,
                transitionName,
                0,
                probability,
                delayGenerator);
    }
    public Transition(long transitionId, String transitionName, int
priority) {
        this(transitionId, transitionName, priority, 1d);
    }
```

```
    public Transition(long transitionId, String transitionName, double
probability) {
        this(transitionId, transitionName, 0, probability);
    }
    public Transition(long transitionId, String transitionName,
DelayGenerator delayGenerator, boolean collectStatistics) {
        this(transitionId, transitionName, 0, delayGenerator,
collectStatistics);
    }
    public Transition(long transitionId, String transitionName,
DelayGenerator delayGenerator) {
        this(transitionId, transitionName, 0, delayGenerator);
    }
    public Transition(long transitionId, String transitionName) {
        this(transitionId, transitionName, 0);
    }
    public void addArcFrom(Place place, int multiplicity) {
        arcsIn.add(new ArcIn(place, this, multiplicity));
    }
    public void addArcFrom(Place place) {
        addArcFrom(place, 1);
    }
    public void addInformationalArcFrom(Place place, int multiplicity) {
        arcsIn.add(new InformationalArcIn(place, this, multiplicity));
    }
    public void addInformationalArcFrom(Place place) {
        addInformationalArcFrom(place, 1);
    }
    public void addArcTo(Place place, int multiplicity) {
        arcsOut.add(new ArcOut(this, place, multiplicity));
    }
    public void addArcTo(Place place) {
        addArcTo(place, 1);
    }
    public long getTransitionId() {
        return transitionId;
    }
    public String getTransitionName() {
        return transitionName;
    }
    public int getPriority() {
        return priority;
    }
    public double getProbability() {
        return probability;
    }
    public DelayGenerator getDelayGenerator() {
        return delayGenerator;
    }
    public List<ArcIn> getArcsIn() {
        return arcsIn;
    }
    public List<ArcOut> getArcsOut() {
        return arcsOut;
    }
    public PetriObject getPetriObject() {
        return petriObject;
```

```
    }
    public void setPetriObject(PetriObject petriObject) {
        if (this.petriObject != null) {
            throw new IllegalStateException("Petri object is already
set. Probably you are trying to add one transition" +
                    " to two different Petri objects. Transition:\n" +
this);
        }
        this.petriObject = petriObject;
    }
    public void resetPetriObject() {
        petriObject = null;
    }
    public boolean isCollectStatistics() {
        return collectStatistics;
    }
    public void validate() throws PetriObjectModelException {
        if (arcsIn.isEmpty()) {
            throw new PetriObjectModelException("Transition: " + this +
" has no arcs in");
        }
        if (arcsOut.isEmpty()) {
            throw new PetriObjectModelException("Transition: " + this +
" has no arcs out");
        }
        boolean hasNotInformationalArcIn = false;
        for (ArcIn arcIn : arcsIn) {
            if (!(arcIn instanceof InformationalArcIn)) {
                hasNotInformationalArcIn = true;
            }
        }
        if (!hasNotInformationalArcIn) {
            throw new PetriObjectModelException("Transition: " + this +
" has informational arc in, but no not informational arc in");
        }
    }
    public String toString() {
        return "Transition{" +
                "transitionId=" + transitionId +
                ", transitionName='" + transitionName + '\'' +
                ", priority=" + priority +
                ", probability=" + probability +
                '}';
    }
}
```

Клас Place.

```
package edu.pedorenko.petri_object_model.model.place;
import edu.pedorenko.petri_object_model.model.PetriObject;
public class Place {
    private long placeId;
    private String placeName;
    private int marking;
    private PetriObject petriObject;
    private boolean collectStatistics;
```

```
    public Place(long placeId, String placeName, int marking, boolean
collectStatistics) {
        if (marking < 0) {
            throw new IllegalArgumentException("Illegal argument marking
= " + marking + "\n" +
                    "Place marking can't be less then 0.");
        }
        this.placeId = placeId;
        this.placeName = placeName;
        this.marking = marking;
        this.collectStatistics = collectStatistics;
    }
    public Place(long placeId, String placeName, int marking) {
        this(placeId, placeName, marking, true);
    }
    public Place(long placeId, String placeName, boolean
collectStatistics) {
        this(placeId, placeName, 0, collectStatistics);
    }
    public Place(long placeId, String placeName) {
        this(placeId, placeName, 0, true);
    }
    public long getPlaceId() {
        return placeId;
    }
    public String getPlaceName() {
        return placeName;
    }
    public int getMarking() {
        return marking;
    }
    public PetriObject getPetriObject() {
        return petriObject;
    }
    public void setPetriObject(PetriObject petriObject) {
        if (this.petriObject != null) {
            throw new IllegalStateException("Petri object is already
set. Probably you are trying to add one place to" +
                    " two different Petri objects. Place:\n" + this);
        }
        this.petriObject = petriObject;
    }
    public void resetPetriObject() {
        this.petriObject = null;
    }
    public boolean isCollectStatistics() {
        return collectStatistics;
    }
    public String toString() {
        return "Place{" +
                "placeId=" + placeId +
                ", placeName='" + placeName + '\'' +
                ", marking=" + marking +
                '}';
    }
}
```

Клас Arc.

```
package edu.pedorenko.petri_object_model.model.arc;
import edu.pedorenko.petri_object_model.model.place.Place;
import edu.pedorenko.petri_object_model.model.transition.Transition;
public abstract class Arc {
    private String arcId;
    private Place place;
    private Transition transition;
    private int multiplicity;
    protected Arc(String arcId, Place place, Transition transition, int
multiplicity) {
        if (multiplicity <= 0) {
            throw new IllegalArgumentException("Illegal argument
multiplicity = " + multiplicity + "\n" +
                    "Arc multiplicity should be more then 0.");
        }
        this.arcId = arcId;
        this.place = place;
        this.transition = transition;
        this.multiplicity = multiplicity;
    }
    public String getArcId() {
        return arcId;
    }
    public Place getPlace() {
        return place;
    }
    public Transition getTransition() {
        return transition;
    }
    public int getMultiplicity() {
        return multiplicity;
    }
    public String toString() {
        return "Arc{" +
                "arcId='" + arcId + '\'' +
                ", place=" + place +
                ", transition=" + transition +
                ", multiplicity=" + multiplicity +
                '}';
    }
}
```

Клас ArcIn.

```
package edu.pedorenko.petri_object_model.model.arc;
import edu.pedorenko.petri_object_model.model.place.Place;
import edu.pedorenko.petri_object_model.model.transition.Transition;
public class ArcIn extends Arc {
    public ArcIn(Place place, Transition transition, int multiplicity) {
        super(place.getPlaceName() + "|" +
transition.getTransitionName(), place, transition, multiplicity);
    }
}
```

Клас ArcOut.

```
package edu.pedorenko.petri_object_model.model.arc;
import edu.pedorenko.petri_object_model.model.place.Place;
import edu.pedorenko.petri_object_model.model.transition.Transition;
public class ArcOut extends Arc {
    public ArcOut(Transition transition, Place place, int multiplicity)
{
        super(transition.getTransitionName() + "|" +
place.getPlaceName(), place, transition, multiplicity);
    }
}
```

Клас InformationalArcIn.

```
package edu.pedorenko.petri_object_model.model.arc;
import edu.pedorenko.petri_object_model.model.place.Place;
import edu.pedorenko.petri_object_model.model.transition.Transition;
public class InformationalArcIn extends ArcIn {
    public InformationalArcIn(Place place, Transition transition, int
multiplicity) {
        super(place, transition, multiplicity);
    }
}
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2019 р.

Бібліотека паралельних обчислень Петрі-об’єктних моделей

Технічне завдання

КПІ.ІП-5117.045490.03.91

“ПОГОДЖЕНО”

Керівник проекту:

_____ І.В. Стеценко

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____А.В. Педоренко

Київ – 2019 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки:

Галузь застосування:

Наведене технічне завдання поширюється на розробку бібліотеки паралельного обчислення Петрі-об’єктних моделей, котра використовується для імітації систем представлених у вигляді Петрі-об’єктної моделі та призначена для галузей, які мають процеси, які необхідно розробляти, контролювати, оптимізувати чи аналізувати, як, наприклад, комп’ютерні мережі, дизайн транспортних шляхів, обслуговування клієнтів тощо.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки бібліотеки паралельних обчислень Петрі-об’єктних моделей є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації і управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім.Ігоря Сікорського).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для зменшення часу імітацій, які використовуються при розробці, аналізі та оптимізації різноманітних продуктів та бізнес-процесів.

Метою розробки є підвищення швидкодії алгоритму імітації Петрі-об’єктної моделі та представлення цього у вигляді лаконічної гнучкої бібліотеки.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

– послідовна та паралельна імітація Петрі-об’єктних моделей з багатоканальними переходами, конфліктними переходами, багатократними дугами, інформаційними вхідними дугами, константними затримками в переходах, з випадковими затримками в переходах розподіленими рівномірно на відрізку, за експоненціальним та за нормальним розподілом;

– валідація коректності створеної користувачем Петрі-об’єктної моделі;

– надання користувачеві протоколу подій, у якому передається інформація про всі події, які виникають у Петрі-об’єктній моделі, зміна стану її елементів, на яких виникають події та час події;

– надання користувачеві статистики стану елементів моделі, які він позначив як ті, що вимагають статистики.

4.1.2 Розробку виконати на платформі Java.

4.1.3 Додаткові вимоги:

– паралелізм має пришвидшити імітацію хоча б у півтора рази;

– має бути виокремлений модуль з об’єктною моделлю, яку має бути легко створити, наприклад, з JSON, та навпаки перетворити у якийсь формат. Стан цієї моделі не має змінюватися під час імітації, для цього необхідно виділити окремий модуль;

– знаходження компонентів сильної зв’язності Петрі-об’єктів при паралельній імітації має бути реалізовано алгоритмом Косараджу.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

4.2.2 Передбачити захист від некоректних дій користувача.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

4.4.2 Мінімальна конфігурація технічних засобів:

Процесор: Pentium 3 1000MHz
Об‘єм ОЗП: 256 Мб.

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1 Програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN32 (Windows'XP, Windows NT і т.д.) та Unix.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Згенерувати установчу версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

5.2 Програмне забезпечення повинно мати довідникову систему

5.3 У склад супроводжувальної документації повинні входити наступні документи:

5.3.1 Пояснювальна записка не менше ніж на 60 аркушах формату А4 (без додатків 5.3.2 - 5.3.6).

5.3.2 Технічне завдання.

5.3.3 Керівництво користувача.

5.3.4 Програма та методика тестування

5.4 Графічна частина повинна бути виконана на трьох аркушах формату А3, котрі включаються у якості додатків до пояснювальної записки:

5.4.1 Схема функціональна програмного забезпечення.

5.4.2 Схема структурна компонент.

5.4.3 Схема структурна класів програмного забезпечення

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

	Назва етапу	Строк	Звітність
1	Вивчення літератури за тематикою проекту	28.04.19	
2	Розробка технічного завдання	05.05.19	Технічне завдання
3	Аналіз вимог та уточнення специфікацій	05.05.19	Специфікації програмного забезпечення
4	Проектування структури програмного забезпечення, проектування компонентів	12.05.19	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5	Програмна реалізація програмного забезпечення	19.05.19	Тексти програмного забезпечення
6	Тестування програмного забезпечення	26.05.19	Тести, результати тестування
7	Розробка матеріалів текстової частини проекту	26.05.19	Пояснювальна записка.
8	Розробка матеріалів графічної частини проекту	28.05.19	Графічний матеріал проекту
9	Оформлення технічної документації проекту	28.05.19	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1 Види випробувань

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2019 р.

Бібліотека паралельних обчислень Петрі-об’єктних моделей

Програма та методика тестування

КПІ.ІП-5117.045490.04.51

“ПОГОДЖЕНО”

Керівник проекту:

_____ І.В. Стеценко

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____А.В. Педоренко

Київ – 2019 року

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Бібліотека паралельної імітації Петрі-об’єктних моделей, яка являє собою jar-файл, створений з використанням мови програмування Java та технології ForkJoinPool.

2 МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

- вібповідність функціоналу вимогам Технічного завдання;
- зручність роботи з бібліотекою.

3 МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Black Box Testing. Перевіряється безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

- функціональне тестування;
- навантажувальне тестування.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується засобами JUnit для модульного тестування.

Працездатність бібліотеки перевіряється шляхом:

– динамічного ручного тестування на відповідність функціональним вимогам;

– статичного тестування коду;

– тестування при максимальному навантаженні;

– тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”
В.о. завідувача кафедри
_____ О.А. Павлов
“___” _____ 2019 р.

Бібліотека паралельних обчислень Петрі-об’єктних моделей

Керівництво користувача

КПІ.ІП-5117.045490.05.34

“ПОГОДЖЕНО”
Керівник проекту:
_____ І.В. Стеценко

Нормоконтроль:
_____ К.І. Ліщук

Виконавець:
_____А.В. Педоренко

Київ – 2019 року

ЗМІСТ

1 ЗАГАЛЬНІ ВІДОМОСТІ

Бібліотека паралельних обчислень Петрі-об’єктних моделей – це готове рішення для оптимізації процесу імітації Петрі-об’єктних моделей. Бібліотека дає можливість запускати імітацію послідовно або паралельно та отримувати необхідну інформацію про події під час імітації разом зі статистикою моделі.

Бібліотека підтримує імітацію Петрі-об’єктних моделей з багатоканальними переходами, затримки у яких вказані у вигляді константи, або розподілені рівномірно на відрізку чи за експоненціальним або нормальним законами. Багатократні дуги та інформаційні вхідні дуги також підтримуються бібліотекою.

2 ПІДГОТОВКА ДО РОБОТИ

Для використання бібліотеки паралельної імітації Петрі-об’єктних моделей jar-файл необхідно спочатку імпортувати у програмний проект. Щоб імпортувати бібліотеку у проект InteliJ IDEA потрібно виконати наступні кроки:

а) натиснути на меню “Файл”;

б) обрати з контекстного меню варіант “Структура проекту”;

в) натиснути на напис “Модулі” на лівій панелі;

г) перейти на вкладку “Залежності”;

д) натиснути на “+”;

е) обрати з контекстного меню варіант “Jar-файли або теки”;

ж) знайти jar-файл з бібліотекою на файловій системі та натиснути “Відкрити”.

Для імпорту бібліотеки у проект Eclipse дотримуйтесь таких кроків:

а) натиснути правою кнопкою миші на проект на лівій панелі;

б) навести мишкою на варіант “Шлях складання” в контекстному меню;

в) обрати з контекстного меню варіант “Налаштувати шлях складання”;

г) перейти на вкладку “Бібліотеки”;

д) натиснути на кнопку “Додати зовнішні jar-файли”;

е) знайти jar-файл з бібліотекою на файловій системі та натиснути “Відкрити”.

3 РОБОТА З БІБЛІОТЕКОЮ

3.1 Створення об’єктної моделі Петрі-об’єктної моделі

Об’єктна модель Петрі-об’єктної моделі представлена класом PetriObjectModel. Конструктор класу PetriObjectModel має лише один параметр – petriObjects. Параметр petriObjects має тим List<PetriObject>. Параметр petriObjects – список об’єктних моделей Петрі-об’єктів.

Об’єктна модель Петрі-об’єкта представлена класом PetriObject, який має декілька конструкторів, так як деякі параметри Петрі-об’єкта опціональні. Параметри об’єктної моделі Петрі-об’єкта:

– petriObjectId – унікальний ідентифікатор Петрі-об’єкта. Тип даних – long;

– petriObjectName - назва Петрі-об’єкта. Тип даних – String;

– transitions - список об’єктних моделей переходів цього Петрі-об’єкта. Тип даних – List<Transition>;

– priority — опціональне значення пріоритету запуска цього Петрі-об’єкта при виникненні конфлікту. Тип даних – int. Значення за замовчуванням – 1;

– externalBufferSizeLimit – опціональне значення обмеження розміру мінімального буфера вихідних подій. Тип даних – int. Значення за замовчуванням – 2147483647.

Об’єктна модель переходу представлена класом Transition який має декілька конструкторів, так як деякі параметри переходу опціональні. Параметри об’єктної моделі переходу:

– transitionId – унікальний ідентифікатор переходу. Тип даних – long;

– transitionName – назва переходу. Тип даних – String;

– priority - опціональне значення пріоритету запуска цього переходу при виникненні конфлікту. Тип даних – int. Значення за замовчуванням – 1;

– probability - опціональне значення ймовірності запуска цього переходу при виникненні конфлікту. Тип даних – double. Значення за замовчуванням – 1;

– delayGenerator – генератор затримки в переході. Тип даних – DelayGenerator;

– collectStatistics – опціональний флаг, який показує чи потрібно збирати статистику для цього переходу. Тип даних – boolean. Значення за замовчуванням – true.

Об’єктна модель позиції представлена класом Place який має декілька конструкторів, так як деякі параметри позиції опціональні. Параметри об’єктної моделі позиції:

– placeId – унікальний ідентифікатор позиції. Тип даних – long;

– placeName – назва позиції. Тип даних – String;

– marking – опціональне значення початкового маркування позиції. Тип даних – int. Значення за замовчуванням – 0.

– collectStatistics – опціональний флаг, який показує чи потрібно збирати статистику для цієї позиції. Тип даних – boolean. Значення за замовчуванням – true.

Для того, щоб створити об’єктну моделі Петрі-об’єктної моделі потрібно спочатку створити позиції для кожного Петрі-об’єкта за допомогою їх конструкторів, параметри яких наведено вище. За тим необхідно створити переходи для кожного Петрі-об’єкта за допомогою їх конструкторів. Після цієї стації кожен перехід має бути об’єднаний з потрібними позиціями в середині одного Петрі-об’єкта. Об’єднання виконується за допомогою методів addArcFrom, addAcrTo та addInformationalArcTo на об’єкті Transition, які додають до переходу вхідну, вихідну та вхідну інформаційну дугу відповідно. Ці методи приймають параметром позицію, з якою поєднується перехід, та опціонально кратність дуги представлену у вигляді типу даних int. Отримані переходи потрібно зібрати у списки по одному на Петрі-об’єкт та створити ці

Петрі-об’єкти за допомогою їх конструкторів, параметри яких наведено вище. Приклад створення об’єктної моделі Петрі-об’єкта наведено на рисунку 3.1

```
Place p3 = new Place( placeId: 3,  placeName: "P3",  collectStatistics: false);
Place p5 = new Place( placeId: 5,  placeName: "P5",  collectStatistics: false);
Place p6 = new Place( placeId: 6,  placeName: "P6",  collectStatistics: false);
Place p7 = new Place( placeId: 7,  placeName: "P7",  collectStatistics: true);
Transition t2 = new Transition( transitionId: 2,  transitionName: "T2", new NormalDelayGenerator( meanDelay: 3,  delayDeviation: 1));
t2.addArcFrom(p5);
t2.addArcFrom(p3);
t2.addArcFrom(p6);
t2.addArcTo(p7);
return new PetriObject( petriObjectId: 2,  petriObjectName: "Object 2", new ArrayList<Transition>(){{add(t2);}});
```

Рисунок 3.1 – Приклад створення об’єктної моделі Петрі-об’єкта

Після створення Петрі-об’єктів можна об’єднувати позиції і переходи, які належать до різних Петрі-об’єктів за допомогою тих же методів, описаних вище. Щоб об’єднати позиції та переходи, які належать до різних Петрі-об’єктів, може знадобитися отримання їх об’єктних моделей з об’єктної моделі Петрі-об’єкта. Для цього клас PetriObject має методи getPlaceById та getTransitionById, які приймають на вхід унікальні ідентифікатори позиції або переходу відповідно. Після того, як усі зв’язки налаштовано, об’єктні моделі Петрі-об’єктів необхідно зібрати у список та створити об’єктну модель Петрі-об'єктну моделі за допомогою її конструктора. Приклад створення об’єктної моделі Петрі-об’єктної моделі наведено на рисунку 3.2.

```
PetriObject petriObject1 = createPetriObject1();

PetriObject petriObject2 = createPetriObject2();

PetriObject petriObject3 = createPetriObject3();

PetriObject petriObject4 = createPetriObject4();

List<PetriObject> petriObjectList = new ArrayList<>();
petriObjectList.add(petriObject1);
petriObjectList.add(petriObject2);
petriObjectList.add(petriObject3);
petriObjectList.add(petriObject4);

petriObject1.getTransitionById( transitionId: 1).addArcTo(petriObject4.getPlaceById( placeId: 4),  multiplicity: 7);
petriObject1.getTransitionById( transitionId: 1).addArcTo(petriObject3.getPlaceById( placeId: 2));
petriObject1.getTransitionById( transitionId: 1).addArcTo(petriObject2.getPlaceById( placeId: 3));

petriObject3.getTransitionById( transitionId: 3).addArcTo(petriObject2.getPlaceById( placeId: 5));
petriObject4.getTransitionById( transitionId: 4).addArcTo(petriObject2.getPlaceById( placeId: 6));

PetriObjectModel petriObjectModel = new PetriObjectModel(petriObjectList);
```

Рисунок 3.2 – Приклад створення об’єктної моделі Петрі-об’єктної моделі

3.2 Послідовна імітація Петрі-об’єктної моделі

Для запуску послідовної імітації Петрі-об’єктної моделі її необхідно передати у статичний метод computeSequential класу PetriObjectModelComputer. Крім об’єктної моделі Петрі-об’єктної моделі метод вимагає на вхід час імітації у вигляді double значення та реалізацію інтерфейсу EventProtocol, яку має створити користувач. Інтерфейс EventProtocol потрібен для того, щоб користувач міг отримувати повідомлення з інформацією про події імітації. Під кожен тип події виділений окремий метод, який отримує своє повідомлення. Окремо необхідно виділити метод isEnabled, який повинен повертати булевий флаг, що показує, чи потрібно надсилати повідомлення про імітацію. Імітація працює швидше, якщо повідомлення надсилати не потрібно. Приклад реалізації інтерфейсу EventProtocol наведено на рисунку 3.3.

```
public class DummyEventProtocol implements EventProtocol {
    public DummyEventProtocol() {
    }
    public boolean isEnabled() { return false; }
    public void onTimeMovedEvent(TimeMovedEvent timeMovedEvent) {
    }
    public void onTransitionActInCompletedEvent(TransitionActInCompletedEvent transitionActInCompletedEvent) {
    }
    public void onPetriObjectActOutCompletedEvent(PetriObjectActOutCompletedEvent petriObjectActOutCompletedEvent) {
    }
    public void onPetriObjectActInCompletedEvent(PetriObjectActInCompletedEvent petriObjectActInCompletedEvent) {
    }
    public void onPetriObjectModelActOutCompletedEvent(PetriObjectModelActOutCompletedEvent petriObjectModelActOutCompletedEvent) {
    }
    public void onPetriObjectModelActInCompletedEvent(PetriObjectModelActInCompletedEvent petriObjectModelActInCompletedEvent) {
    }
    public void onPetriObjectsConflictResolvedEvent(PetriObjectsConflictResolvedEvent petriObjectsConflictResolvedEvent) {
    }
    public void onTransitionsConflictResolvedEvent(TransitionsConflictResolvedEvent transitionsConflictResolvedEvent) {
    }
}
```

Рисунок 3.3 – Приклад реалізації інтерфейсу EventProtocol

3.3 Паралельна імітація Петрі-об’єктної моделі

Для запуску паралельної імітації Петрі-об’єктної моделі її необхідно передати у статичний метод computeParallel класу PetriObjectModelComputer. Крім об’єктної моделі Петрі-об’єктної моделі метод вимагає на вхід час імітації у вигляді double значення та реалізацію інтерфейсу EventProtocolFactory, яку має створити користувач. Реалізація EventProtocolFactory має повертати новий об’єкт реалізації інтерфейсу EventProtocol. Окремий EventProtocol буде використовуватися в кожному Петрі-об’єкті для того, щоб не алгоритм працював швидше. Приклад реалізації EventProtocolFactory наведено на рисунку 3.4.

```
public class DummyEventProtocolFactory implements EventProtocolFactory {

    public DummyEventProtocolFactory() {
    }

    public EventProtocol createEventProtocol() { return new DummyEventProtocol(); }
}
```

Рисунок 3.4 – Приклад реалізації інтерфейсу EventProtocolFactory

Приклад запуску імітації паралельно та послідовно наведено на рисунку 3.5.

```
PetriObjectModelStatistics statisticsSequential = PetriObjectModelComputer.computeSequential(petriObjectModel, timeModeling: 100000, new DummyEventProtocol());
PetriObjectModelStatistics statisticsParallel = PetriObjectModelComputer.computeParallel(petriObjectModel, timeModeling: 100000, new DummyEventProtocolFactory());
```

Рисунок 3.5 – Приклад запуску імітації паралельно та послідовно

3.4 Отримання статистики імітації

Методи класу PetriObjectModelComputer для послідовної та паралельної імітації повертають об’єктну модель статистики імітації Петрі-об’єктної моделі. Там знаходиться статистика позицій та переходів, для яких флаг collectStatistics був встановлений у true при створенні. Щоб отримати статистику конкретної позиції або переходу необхідно зі статистики Петрі-об’єктної моделі за допомогою метода getPetriObjectStatisticsByPetriObjectId дістати статистику Петрі-об’єкта за чисельним ідентифікатором цього Петрі-об’єкта, а потім за допомогою метода getTransitionStatisticsByTransitionId чи

getPlaceStatisticsByPlaceId за чисельним ідентифікатором переходу чи позиції отримати зі статистики Петрі-об’єкта статистику переходу чи позиції відповідно.

Статистика переходу містить в собі інформацію про кількість подій в переході на момент завершення імітації, максимальну, мінімальну та середню кількість подій в переході за весь час імітації, яку можна отримати відповідними методами.

Статистика позиції містить в собі інформацію про маркування позиції на момент завершення імітації, максимальне, мінімальне та середнє маркування позиції за весь час імітації, яку можна отримати відповідними методами.

Приклад роботи зі статистикою наведено на рисунку 3.6.

```
System.out.println("Sequential statistics");
System.out.println("Place P7: " + statisticsSequential.getPetriObjectStatisticsByPetriObjectId(2).getPlaceStatisticsByPlaceId(7));
System.out.println();
System.out.println("Parallel statistics");
System.out.println("Place P7: " + statisticsParallel.getPetriObjectStatisticsByPetriObjectId(2).getPlaceStatisticsByPlaceId(7));
```

Рисунок 3.6 – Приклад роботи зі статистикою

4 РЕКОМЕНДАЦІЇ ПО ОСВОЄННЮ

Для кращого розуміння роботи з бібліотекою, необхідно мати досвід роботи із Java, багатопоточністю, розуміти алгоритм імітації Петрі-об’єктних моделей та ознайомитися з даним керівництвом користувача.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2019 р.

Бібліотека паралельних обчислень Петрі-об’єктних моделей

Графічні матеріали

КПІ.ІП-5117.045490.06.99

“ПОГОДЖЕНО”

Керівник проекту:

_____ І.В. Стеценко

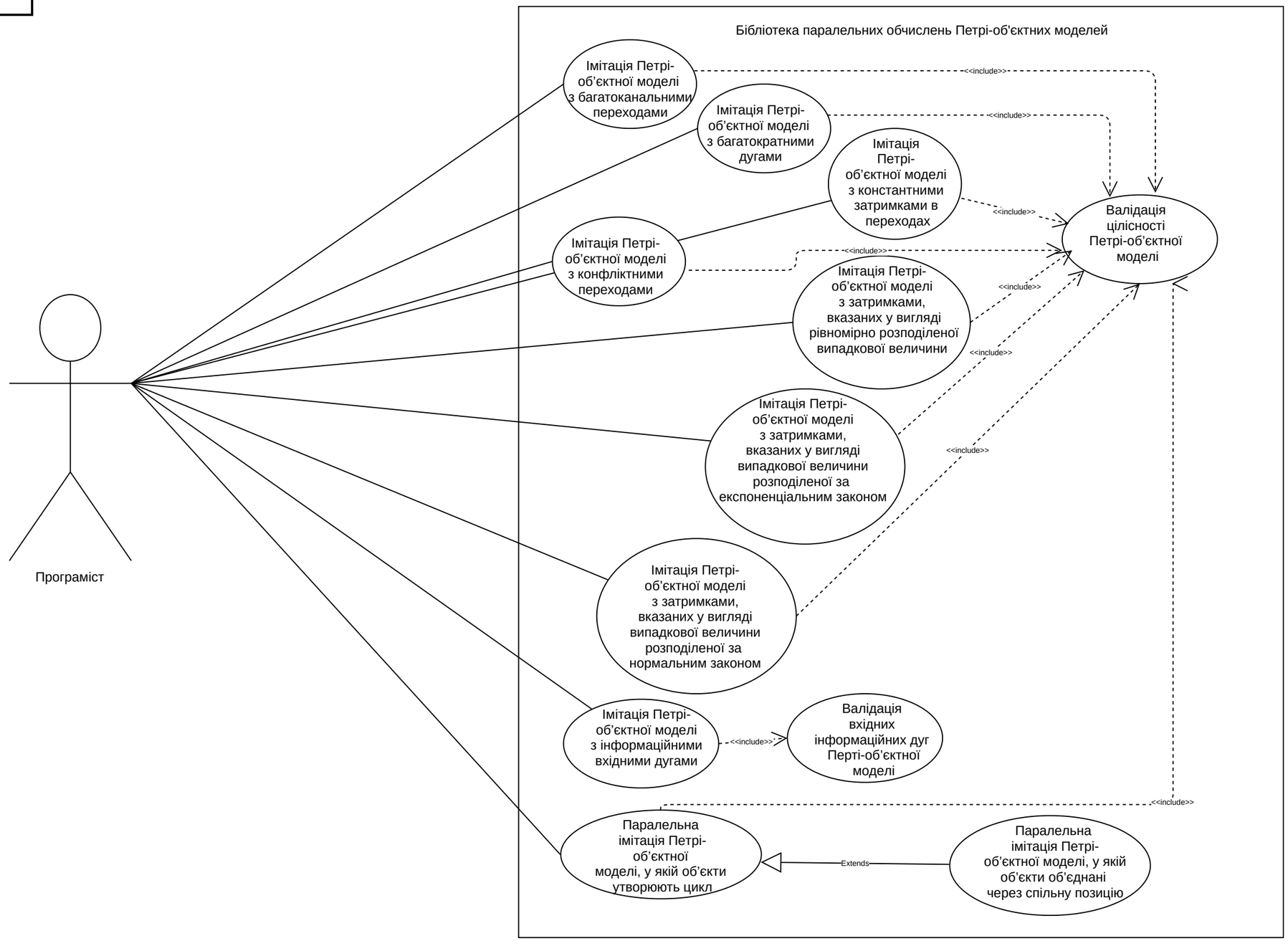
Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

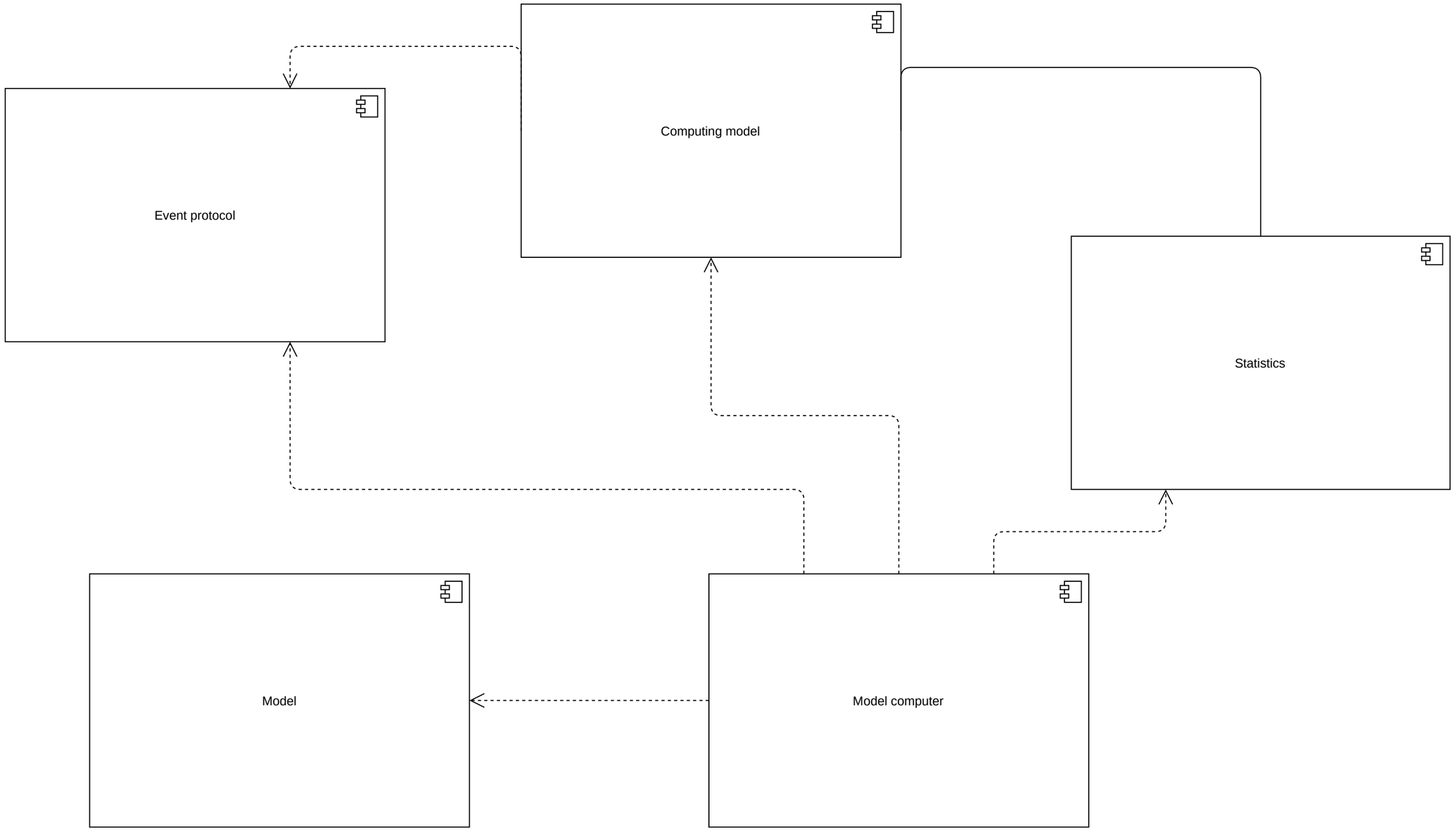
_____А.В. Педоренко

Київ – 2019 року



					КПІ.ІП-5117.045490.06.99.СС		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використань	Літера	Маса / Масштаб
Розробив		Педоренко А.В.					
Перевірив		Стеценко І.В.				Аркуш 1	Аркушів 1
Т. кон.							
Н. кон.		Ліщук К.І.			Бібліотека паралельних обчислень Петрі-об'єктних моделей	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІП-51	
Затвердив		Стеценко І.В.					

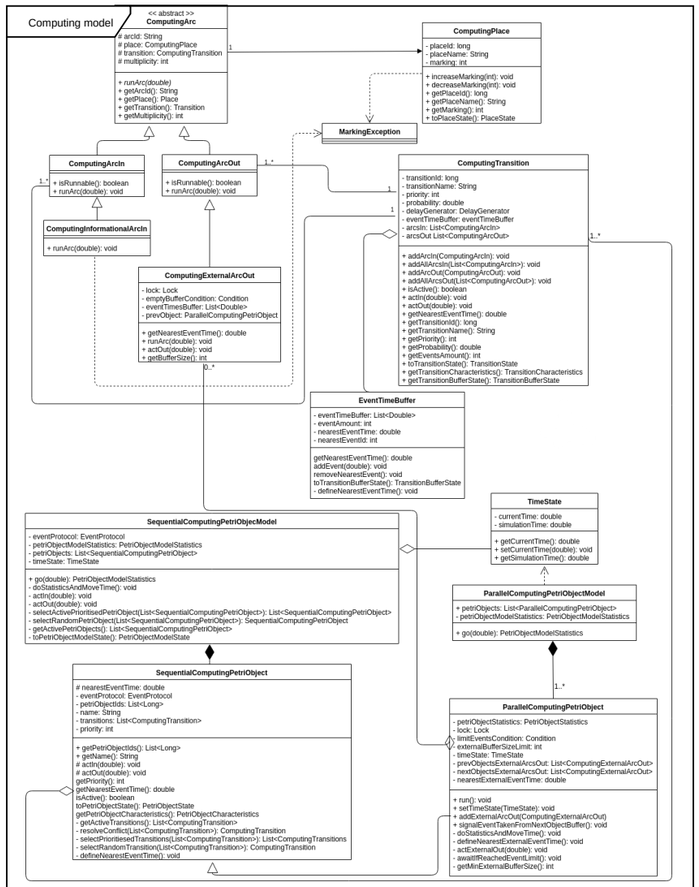
КПІ.ІП-5117.045490.06.99.СС



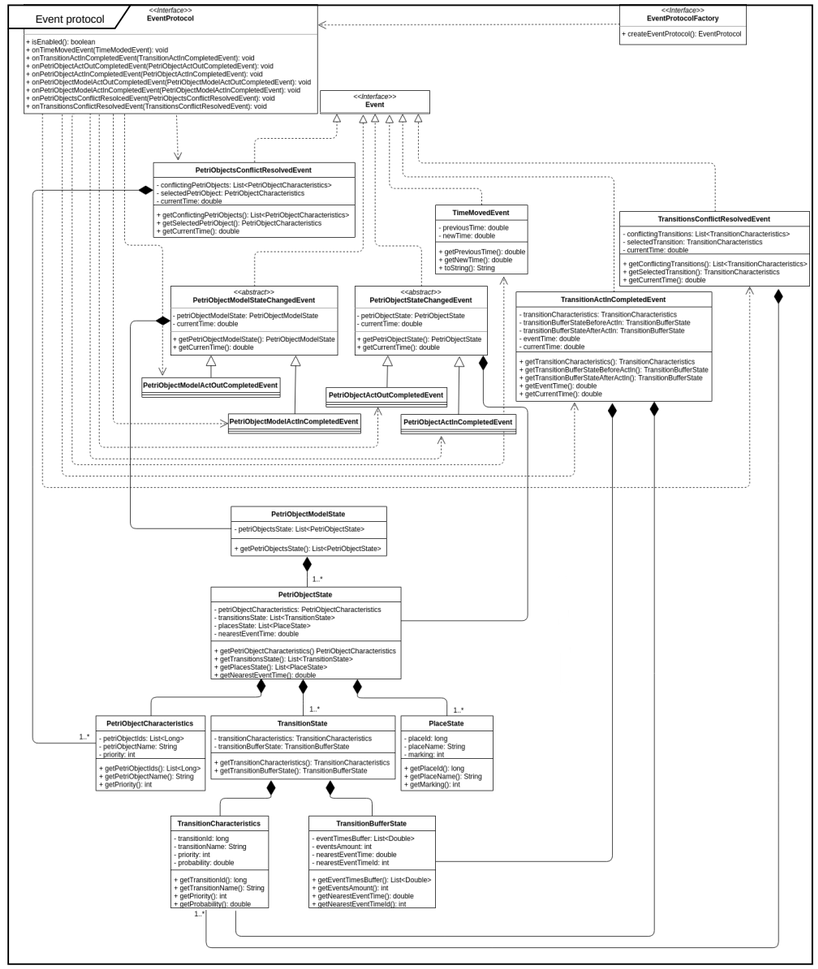
					КПІ.ІП-5117.045490.06.99.СС	
					Схема структурна компонентів програмного забезпечення	Літера / Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата		
Розробив		Педоренко А.В.				Аркуш 1 / Аркушів 1
Перевірив		Стеценко І.В.				
Т. кон.						
Н. кон.		Ліщук К.І.			Бібліотека паралельних обчислень Петрі-об'єктних моделей	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІП-51
Затвердив		Стеценко І.В.				

КПІ.ІП-5117.045490.06.99.СС

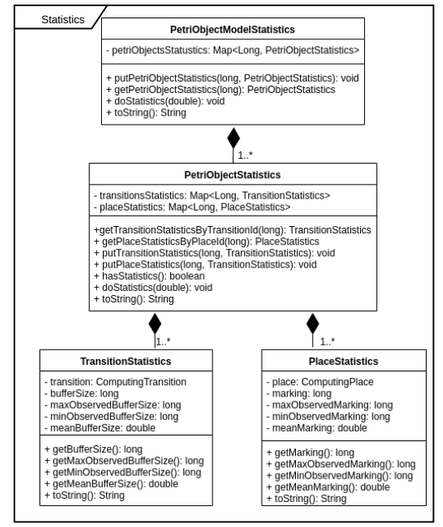
Computing model



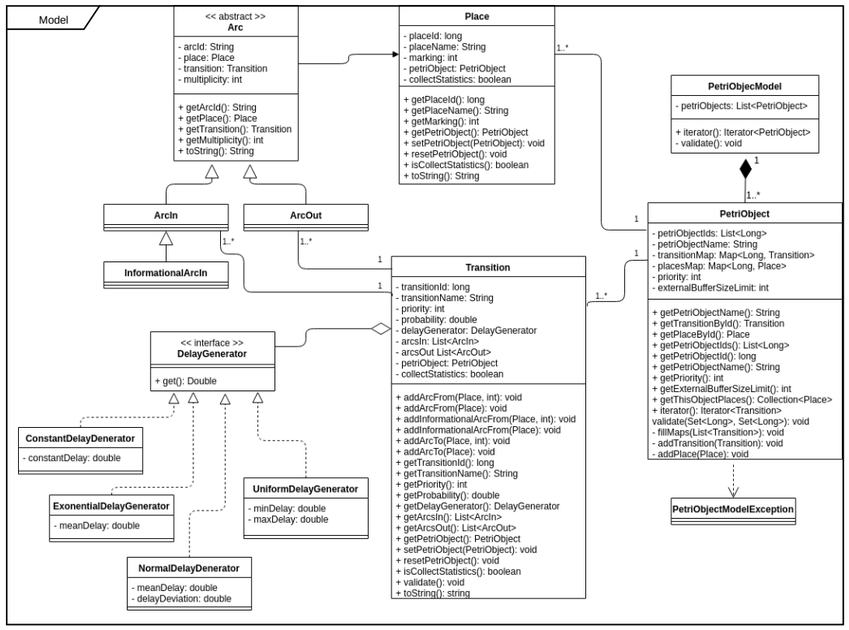
Event protocol



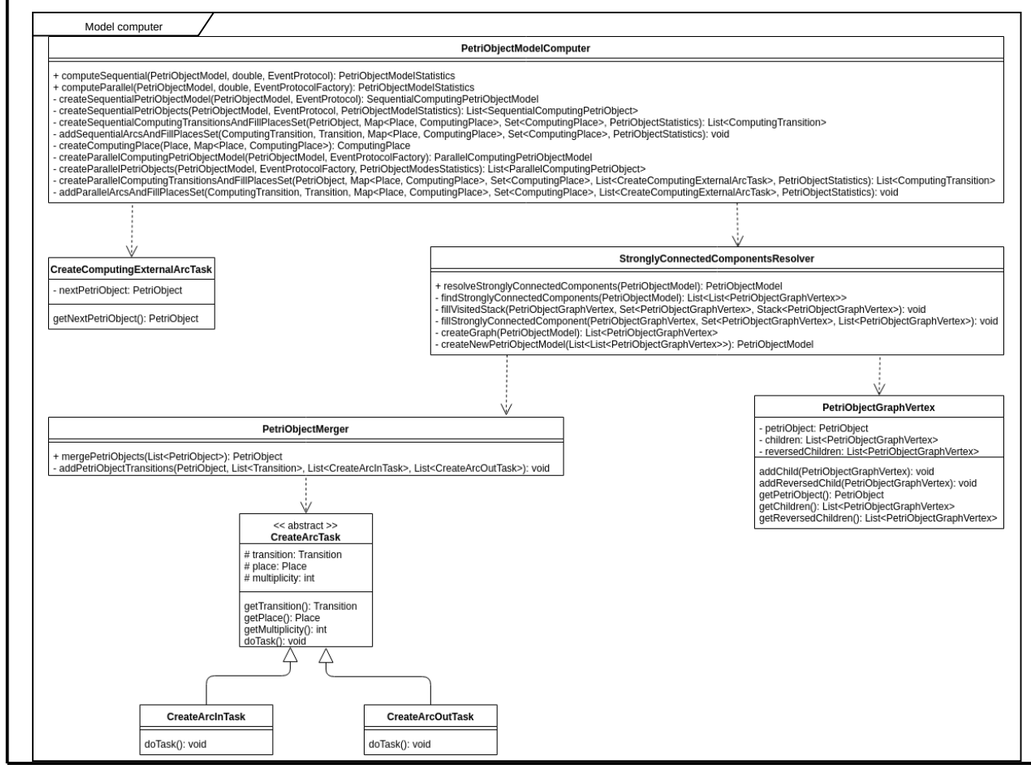
Statistics



Model



Model computer



					КПІ.ІП-5117.045490.06.99.СС	
Зм.	Арк.	№ документа	Підпис	Дата		
Розробив		Педоренко А.В.			Схема структурна класів програмного забезпечення	Літера / Маса / Масштаб
Перевірив		Стеценко І.В.				Аркуш 1 / Аркушів 1
Т. кон.						
Н. кон.		Ліщук К.І.			Бібліотека паралельних обчислень Петрі-об'єктних моделей	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІП-51
Затвердив		Стеценко І.В.				